

# BlackBerry Browser

Version 4.2

Content Developer Guide

## BlackBerry Browser Version 4.2 Content Developer Guide

Last modified: 10 April 2007

Document number: 10635058

At the time of publication, this documentation complies with BlackBerry Device Software Version 4.2.

©2007 Research In Motion Limited. All Rights Reserved. The BlackBerry and RIM families of related marks, images, and symbols are the exclusive properties of Research In Motion Limited. RIM, Research In Motion, BlackBerry, "Always On, Always Connected" and the "envelope in motion" symbol are registered with the U.S. Patent and Trademark Office and may be pending or registered in other countries.

IBM, Lotus, Domino, and Lotus Notes are trademarks of IBM in the United States. Microsoft is a registered trademark of Microsoft Corporation in the United States and/or other countries. Microsoft and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Adobe and Photoshop are registered trademarks of Adobe Systems Incorporated in the United States, and/or other countries. Java and all trademarks and logos that contain Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Mobitex is either a registered trademark or trademark of Telefonaktiebolaget LM Ericsson Corporation. All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

The BlackBerry device and/or associated software are protected by copyright, international treaties, and various patents, including one or more of the following U.S. patents: 6,278,442; 6,271,605; 6,219,694; 6,075,470; 6,073,318; D445,428; D433,460; D416,256. Other patents are registered or pending in various countries around the world. Visit [www.rim.com/patents](http://www.rim.com/patents) for a list of RIM [as hereinafter defined] patents.

This document is provided "as is" and Research In Motion Limited and its affiliated companies ("RIM") assume no responsibility for any typographical, technical, or other inaccuracies in this document. In order to protect RIM proprietary and confidential information and/or trade secrets, this document may describe some aspects of RIM technology in generalized terms. RIM reserves the right to periodically change information that is contained in this document; however, RIM makes no commitment to provide any such changes, updates, enhancements, or other additions to this document to you in a timely manner or at all. RIM MAKES NO REPRESENTATIONS, WARRANTIES, CONDITIONS, OR COVENANTS, EITHER EXPRESS OR IMPLIED (INCLUDING WITHOUT LIMITATION, ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OF FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, MERCHANTABILITY, DURABILITY, TITLE, OR RELATED TO THE PERFORMANCE OR NON-PERFORMANCE OF ANY SOFTWARE REFERENCED HEREIN OR PERFORMANCE OF ANY SERVICES REFERENCED HEREIN). IN CONNECTION WITH YOUR USE OF THIS DOCUMENTATION, NEITHER RIM NOR ITS RESPECTIVE DIRECTORS, OFFICERS, EMPLOYEES, OR CONSULTANTS SHALL BE LIABLE TO YOU FOR ANY DAMAGES WHATSOEVER BE THEY DIRECT, ECONOMIC, COMMERCIAL, SPECIAL, CONSEQUENTIAL, INCIDENTAL, EXEMPLARY, OR INDIRECT DAMAGES, EVEN IF RIM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, INCLUDING WITHOUT LIMITATION, LOSS OF BUSINESS REVENUE OR EARNINGS, LOST DATA, DAMAGES CAUSED BY DELAYS, LOST PROFITS, OR A FAILURE TO REALIZE EXPECTED SAVINGS.

This document might contain references to third-party sources of information, hardware or software, products or services and/or third-party web sites (collectively the "Third-Party Information"). RIM does not control, and is not responsible for, any Third-Party Information, including, without limitation the content, accuracy, copyright compliance, compatibility, performance, trustworthiness, legality, decency, links, or any other aspect of Third-Party Information. The inclusion of Third-Party Information in this document does not imply endorsement by RIM of the Third-Party Information or the third-party in any way. Installation and use of Third-Party Information with RIM's products and services may require one or more patent, trademark, or copyright licenses in order to avoid infringement of the intellectual property rights of others. Any dealings with Third-Party Information, including, without limitation, compliance with applicable licenses and terms and conditions, are solely between you and the third-party. You are solely responsible for determining whether such third-party licenses are required and are responsible for acquiring any such licenses relating to Third-Party Information. To the extent that such intellectual property licenses may be required, RIM expressly recommends that you do not install or use Third-Party Information until all such applicable licenses have been acquired by you or on your behalf. Your use of Third-Party Information shall be governed by and subject to you agreeing to the terms of the Third-Party Information licenses. Any Third-Party Information that is provided with RIM's products and services is provided "as is". RIM makes no representation, warranty or guarantee whatsoever in relation to the Third-Party Information and RIM assumes no liability whatsoever in relation to the Third-Party Information even if RIM has been advised of the possibility of such damages or can anticipate such damages.

Research In Motion Limited  
295 Phillip Street  
Waterloo, ON N2L 3W8  
Canada

Published in Canada

Research In Motion Europe  
Centrum House, 36 Station Road  
Egham, Surrey TW20 9LF  
United Kingdom



# Contents

<b>1</b>	<b>Getting started with the BlackBerry Browser</b>	<b>9</b>
	Using the BlackBerry Browser	9
	Browser configurations	9
	BlackBerry Browser configuration	11
	Internet Browser configuration	11
	WAP Browser configuration	12
	Browser feature support	14
	Browser content support	17
	Managing multipart content	20
	Determining which markup languages are accepted	20
	Image conversion	20
<b>2</b>	<b>Browser interface and features</b>	<b>23</b>
	Browser screen	23
	Browser menus	23
	WML <do> elements	24
	Links	24
	Option lists	25
	Browser features	25
	History	25
	Cookies	25
	Cache	26
	Bookmarks	26
<b>3</b>	<b>Designing wireless web content for the BlackBerry Browser</b>	<b>29</b>
	Creating effective content for the BlackBerry Browser	29
	Follow basic web design principles	29
	Organize content effectively	29
	Select the most appropriate markup language	30
	Consider BlackBerry device screen sizes	31
	Encourage text entry	31
	Minimize download time	31
	Improve rendering time	31
	Creating effective images	31

Defining queues for offline form submission .....	33
Create an HTTP header property file .....	34
Add queuing parameters directly to the web page .....	34
Making requests for content only when content has changed .....	35
Delivering device-specific content .....	35
Write a browser detection script .....	36
Send device-appropriate images .....	38
<b>4 Creating XHTML pages .....</b>	<b>41</b>
Using XHTML-MP .....	41
Creating XHTML-MP-compliant sites .....	41
Creating an XHTML-MP page .....	43
Code sample: Creating an XHTML-MP web page .....	49
<b>5 Creating WML pages .....</b>	<b>55</b>
Using WML .....	55
WML design tips .....	55
Creating a WML page .....	57
Code sample: Creating a WML web page .....	62
<b>6 Creating browser push applications .....</b>	<b>67</b>
Push applications .....	67
BlackBerry Browser configuration push support .....	67
WAP Browser configuration push support .....	70
The BlackBerry push process .....	71
Defining push attributes .....	71
BlackBerry MDS Connection Service push attributes .....	71
PAP push HTTP header .....	78
Browser push HTTP headers .....	78
RIM push service implementation .....	79
Writing a RIM push service application .....	79
Code sample: Creating a browser push application using the RIM push service implementation .....	81
PAP push service implementation .....	85
Writing a PAP push service application .....	85
Code sample: Creating a browser push application using the PAP push service implementation .....	90

<b>7</b>	<b>Testing web pages</b>	<b>99</b>
	Using the simulators	99
<b>A</b>	<b>XHTML language reference</b>	<b>101</b>
	XHTML-MP reference	101
	Structural elements	101
	Text and text formatting elements	102
	Link elements	105
	List elements	106
	Basic form elements	107
	Basic table elements	108
	Image elements	109
	Object elements	110
	Meta information	110
	Script references	111
	WAP CSS reference	112
	Element and CSS property matrix	120
<b>B</b>	<b>WML language reference</b>	<b>123</b>
	WML reference	123
	Structure elements	123
	Text and text formatting elements	124
	Link elements	125
	Table elements	126
	Image elements	126
	Event elements	127
	Task elements	128
	Input elements	129
	Variable elements	130
<b>C</b>	<b>JavaScript language reference</b>	<b>133</b>
	Using JavaScript	133
	Supported JavaScript objects	133
	BlackBerry	134
	BlackBerry Location	135
	Navigator	138
	Document	142
	Form	149

Screen .....	156
Window .....	158
Window History .....	170
<b>D WMLScript language reference .....</b>	<b>173</b>
Using WMLScript .....	173
WMLScript libraries .....	174
Lang .....	174
Dialogs .....	178
String .....	179
URL .....	185
Browser .....	190
<b>E Scripting Basics .....</b>	<b>193</b>
Reserved words .....	193
Statements .....	194
Operators and expressions .....	196



# Getting started with the BlackBerry Browser

Using the BlackBerry Browser  
Browser configurations  
Browser feature support  
Browser content support

## Using the BlackBerry Browser

The BlackBerry® Browser is designed to let users access and navigate web pages over a wireless connection just as they would using a desktop browser. For content developers, however, wireless browsing poses a number of additional challenges that are not present when designing content for a traditional desktop environment. Some notable differences include:

- **Display size:** The display sizes of the different BlackBerry devices, while not as small as typical wireless devices, are still much smaller than a desktop browser.
- **Memory:** BlackBerry devices have more stringent memory restrictions than desktop computers, which impacts the amount of data they can store.
- **Network:** Wireless networks have considerably slower data transfer rates than standard LAN networks. Most wireless browsers access the Internet through a Wireless Application Protocol (WAP) gateway, which can have size and content limitations.

Research In Motion (RIM) has designed two gateways to mitigate the impact of the wireless network by supporting a wider range of content and optimizing it to reduce content sizes and decrease transmission and rendering times:

- BlackBerry MDS™ Connection Service, part of the BlackBerry® Enterprise Solution
- BlackBerry® Internet Service Browsing, part of the BlackBerry Internet Service retail offering



**Note:** A different browser configuration is available for each of these gateways. See "Browser configurations" on page 9 for more information.

This guide provides an overview of the technical aspects of the BlackBerry Browser and outlines design considerations for different browser configurations.

## Browser configurations

BlackBerry devices can provide up to three browser configurations. Which configurations are provisioned on a BlackBerry device depends in part on how that device connects to a wireless network:

- **BlackBerry Browser:** connects to the wireless network using the BlackBerry MDS Connection Service component of the BlackBerry® Enterprise Server as its gateway

- **Internet Browser:** connects to the wireless network using the BlackBerry Internet Service as its gateway
- **WAP Browser:** connects to the wireless network using a WAP gateway

Each browser configuration represents a different setup of the same browser application. The following table summarizes the characteristics of each configuration:

Configuration	Protocol	Network gateway	Service books	Features
BlackBerry Browser	HTTP/IP Proxy Protocol (IPPP)	BlackBerry MDS Connection Service	<ul style="list-style-type: none"> <li>• MDS Transport</li> <li>• MDS Browser Configuration</li> </ul>	<ul style="list-style-type: none"> <li>• HTTP over SSL/TLS (HTTPS), including secure access to corporate intranets, supports push applications</li> <li>• reads the contents of the browser cache, reduces amount of content sent over the wireless network</li> <li>• content is optimized for wireless transmission and the rendering capabilities of the device before it is sent to the browser</li> </ul>
Internet Browser	HTTP/IP Proxy Protocol (IPPP)	BlackBerry Internet Service	<ul style="list-style-type: none"> <li>• Internet Transport</li> <li>• Internet Browser Configuration</li> </ul>	<ul style="list-style-type: none"> <li>• HTTP over SSL/TLS (HTTPS)</li> <li>• HTTP gateway for network-aware Java™ applications</li> <li>• reads the contents of the browser cache, reduces amount of content sent over the wireless network</li> <li>• content is optimized for wireless transmission and the rendering capabilities of the device before it is sent to the browser</li> </ul>
WAP Browser	WAP 1.2.2 & WAP 2.0	WAP-compliant gateway without proprietary Wireless Markup Language (WML) extensions (must support WAP Transport Protocol-level (WTP) segmentation and reassembly)	<ul style="list-style-type: none"> <li>• WAP Transport</li> <li>• WAP Browser Configuration</li> </ul>	<ul style="list-style-type: none"> <li>• Wireless Transport Layer Security (WTLS)</li> <li>• Wireless Session Protocol (WSP) header caching</li> </ul>

As of BlackBerry Device Software Version 4.2, only one icon for the browser appears on the Home screen. In the event that more than one browser configuration exists on a BlackBerry device, the device users can define the default browser configuration (and, therefore, the default network gateway) that the browser uses.

Users can select the configuration that is optimal for the type of content that they are viewing. For example, users might use the WAP Browser to browse standard WML content on the Internet and use the BlackBerry Browser to access a corporate intranet.

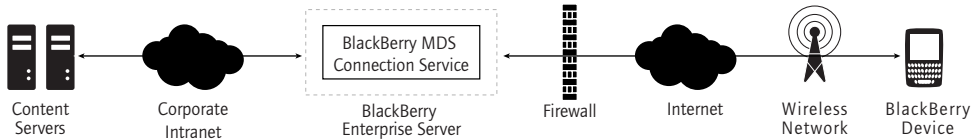
The browser saves bookmarks based on the browser configuration used to view the sites. When the user opens a bookmarked site, the browser automatically uses the associated browser configuration.

Corporate system administrators can set an IT policy on the BlackBerry Enterprise Server to specify the default configuration that the browser uses for URL links in applications other than the browser, such as in email messages or memos. Users can also change the default browser configuration. See the on-device help for more information about changing the default browser configuration.

## BlackBerry Browser configuration

### BlackBerry Browser configuration: Architecture

The BlackBerry Browser is designed to provide corporate customers with secure access to corporate intranets, as well as access to the Internet, using the BlackBerry MDS Connection Service of the BlackBerry Enterprise Server. The BlackBerry Enterprise Server exists on the corporate network.



BlackBerry Browser configuration connection to the network

The BlackBerry Browser communicates over the wireless network using HTTP over the RIM IPPP.

### BlackBerry Browser configuration: Security

Communication between the BlackBerry device and the BlackBerry Enterprise Server is encrypted with Triple Data Encryption Standard (Triple DES).

The BlackBerry Browser configuration supports HTTPS in one of two modes:

- end-to-end mode: HTTP communication is encrypted using SSL or TLS for the entire connection between the BlackBerry device and the originating content server. Communication over the wireless network between the BlackBerry device and the BlackBerry Enterprise Server is also Triple DES-encrypted.



**Note:** To support end-to-end SSL on the BlackBerry device, users must install the optional TLS package. This package is available in BlackBerry® Desktop Manager Version 3.6.1 or later.

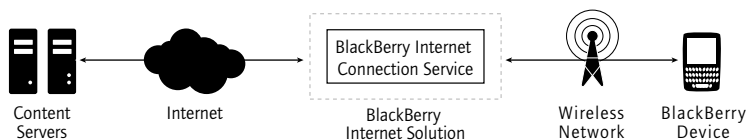
- proxy mode: The BlackBerry Enterprise Server performs SSL handshaking and sets up the SSL connection on behalf of the device. Communication over the wireless network between the BlackBerry device and the BlackBerry Enterprise Server is not encrypted using SSL, but it is still Triple DES-encrypted. Communication over the Internet between the BlackBerry Enterprise Server and the content server is encrypted using SSL or TLS.

## Internet Browser configuration

### Internet Browser configuration: Architecture

The Internet Browser uses the BlackBerry Internet Service as its gateway to the Internet. The BlackBerry Internet Service is a component that exists within the BlackBerry Infrastructure. It contains components that optimize web content for wireless browsing and transcode content types into appropriate formats for display on the device.

Because the BlackBerry Internet Service does not require a BlackBerry Enterprise Server, which is typically exists behind a corporate firewall, it can be made available to non-corporate clients through their BlackBerry service providers.



Internet Browser configuration connection to the network

The Internet Browser communicates over the wireless network using HTTP over the RIM IPPP. Because the RIM IPPP was designed specifically for the BlackBerry Infrastructure, and because the content is preprocessed and optimized for wireless transmission and display on a BlackBerry device, delivery of HTML is both faster and more efficient than HTTP over WAP in most current implementations.

## Internet Browser configuration: Security

Because the Internet Browser is intended for use by prosumers instead of corporate customers, it does not support Triple DES encryption and is not designed to access intranets that are protected by firewalls; however, the Internet Browser can access secure sites using HTTPS and supports SSL encryption.

The Internet Browser configuration supports HTTPS in end-to-end mode only; that is, HTTP communication is encrypted using SSL or TLS for the entire connection between the device and the originating content server.

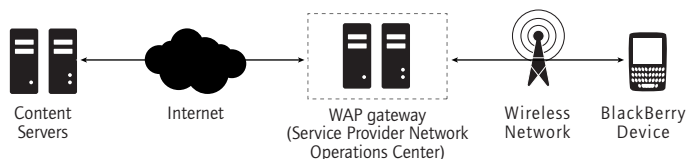


**Note:** To support end-to-end SSL on the device, users must install the optional TLS package. This package is available in BlackBerry Desktop Manager Version 3.6.1 or later.

## WAP Browser configuration

### WAP Browser configuration: Architecture

The WAP Browser connects to the network using a WAP gateway. WAP gateways must support WTP-level segmentation and reassembly. Proprietary WAP extensions are not supported.<sup>1</sup>



WAP Browser configuration connection to the network

The WAP Browser is designed to support WAP 2.0, with the following exceptions:

- no WAP 2.0 provisioning support
- no iCard® support
- no iCalendar® support

<sup>1</sup> Research In Motion does not in any way endorse or guarantee the security, compatibility, performance, or trustworthiness of any WAP gateway, and shall have no liability to you or any third party for issues arising from such WAP gateway.

- limited Wireless Telephony Application Interface (WTAI) support: the browser supports only the Uniform Resource Identifier (URI) forms of the public WTAI functions
- limited WAP cascading style sheet (CSS) support

The WAP Browser configuration supports the following protocols:

- WAP 1.2.1: The WAP Browser caches WSP headers to decrease the transmission time of requests; it sends common HTTP headers to the WAP gateway when it sets up the WAP connection, and then sends only additional or changed headers in each request. In subsequent requests, the WAP Browser sends only headers that are specific to the request or that contain values that are different from the initial values.

See the specification WAP-203-WSP-20000504-a at <http://www.wapforum.org> for more information.

- WAP 2.0: The WAP Browser sends HTTP over Wireless-Profiled TCP (wTCP). The browser sends the HTTP request to a WAP 2.0 proxy, which then forwards it to the server.

The WAP gateway determines which content types the browser can access. For example, some WAP gateways might convert HTML content into a series of WML pages, while some might impose a limit on the size of content that the WAP Browser can request.

### WAP Browser configuration: Security

The WAP Browser is designed to use WTLS to access secure WAP services, including WTLS Class 1 (encryption only, no authentication) and WTLS Class 2 (encryption and server authentication). The WAP Browser configuration supports both DES (40- and 56-bit) and RC5 encryption (64-, 128- and 168-bit). The WAP Browser configuration does not support the WMLScriptCrypto library. Communication over the wireless network between the device and the WAP gateway is encrypted in WTLS. Communication over the Internet between the WAP gateway and the origin web server is encrypted in SSL or TLS.

The WAP gateway decrypts data that it receives from either the device or the originating server and re-encrypts it using the appropriate protocol. During its conversion from one encrypted format to another, data is briefly not encrypted at the service provider location.

To support WTLS on the device, users must install the optional WTLS Security Package for the browser. This package is available for installation in the BlackBerry Desktop Software.

## Browser feature support

Feature	Browser configuration			Description	Supported as of
	BlackBerry	Internet	WAP		
Content transfer and rendering optimization					
Browser Session Management	✓	✓		When a user starts a browser session, the Browser Session Management protocol immediately sends information about the contents of the browser's cache to the network gateway. Using this knowledge of the cache contents, the network gateway can then omit the transfer of any page component (such as an image, style sheet, or JavaScript™ file) that already exists in the cache (and is still valid). The browser revalidates expired components, wherever possible.	v.4.1
Content filtering	✓	✓		The network gateway preprocesses HTML or Extensible HTML (XHTML) content to remove unsupported tags for faster display on the BlackBerry device, converts data to a tokenized format, and compresses the data for efficient delivery over the wireless network. It then sends processed content to the device as soon as it is available.	v.3.7
Page rendering	✓	✓		The network gateway retrieves images from the content server while it preprocesses the HTML or XHTML content. It then includes the images with the pages that it sends to the device for reduced wireless network traffic and faster browsing.	v.3.7
Image optimization	✓	✓		The network gateway converts .jpeg or .gif images to .png images and scales them to fit the screen dimensions. It also reduces the image color depth to suit the capabilities of the user's BlackBerry device.	v.3.6
Mobile-friendly browsing					
Automatic synchronization of bookmarked content	✓	✓	✓	Users can set the browser to automatically update bookmarked content at a specified interval. When the specified interval is reached, the browser will connect to the content server and download the most recent content, if it has been updated since the last time the content was retrieved. If the BlackBerry device is out of wireless coverage when the browser attempts to retrieve the content, the browser will wait until the device is back in coverage, then attempt to retrieve the content again. Content is sent directly to the browser cache. When new content has been cached for a bookmark, the next time the user enters the browser, the bookmark appears in bold and italics.	v.4.2
Offline form submission	✓	✓	✓	When the device is outside of a wireless coverage area, users can fill out an HTML form and have the browser queue it for submission to the server. The browser submits the form as soon as the device is in a wireless coverage area.	v.3.8

Feature	Browser configuration			Description	Supported as of
	BlackBerry	Internet	WAP		
Background downloading	✓	✓	✓	Users can request a page and download it directly to the message list. They can continue to view the current page while the requested page is downloaded in the background. After the page has been completely downloaded, users can go to the message list and select the requested page to view it.	v.3.2
<b>Navigation</b>					
Page overview	✓	✓	✓	Users can zoom out to provide an overview of the entire page so that they can quickly scroll to the area of the page that they want to view.  This allows users to skip over ads or top of page navigation links, instead of using the trackwheel or trackball navigation to scroll over each link before the browser displays the core content.  <b>Note:</b> This feature is not supported for WML content.	v.4.2
Service provider customizable bookmarks	✓	✓	✓	As with desktop browsers, service providers can provision the browser with custom bookmarks.	v.3.8
One-click support for links and image maps	✓	✓	✓	To follow a link, users click and hold the trackwheel or trackball.	v.3.7
Trackwheel or trackball navigation	✓	✓	✓	Using the trackwheel or trackball, users can scroll from link to link. Scrolling up or down moves to the next or previous link on the same line, before moving to the next line.  When images are placed in <a> or <anchor> tags, users can select the image to follow the link or to perform the associated action.	v.3.7
Soft keys for WML <do> items	✓	✓	✓	In addition to being added to the browser menu, WML <do> items are displayed as "soft keys" in a non-scrolling region at the bottom of the browser screen.	v.3.6.1
<b>Security</b>					
SSL/TLS encryption	✓	✓		<p><b>The BlackBerry and Internet Browser configurations support HTTPS in the following way:</b></p> <ul style="list-style-type: none"> <li> <b>BlackBerry Browser:</b> The BlackBerry Browser configuration supports HTTPS in one of two modes: <ul style="list-style-type: none"> <li>end-to-end mode (when the TLS package exists on the BlackBerry device)</li> <li>proxy mode</li> </ul> See "BlackBerry Browser configuration: Security" on page 11 for more information. </li> <li> <b>Internet Browser:</b> When the TLS package exists on the BlackBerry device, the Internet Browser configuration supports HTTPS in end-to-end mode. See "BlackBerry Browser configuration: Security" on page 11 for more information. </li> </ul>	v.3.6

Feature	Browser configuration			Description	Supported as of
	BlackBerry	Internet	WAP		
WTLS encryption			✓	The WAP Browser configuration is designed to support WTLS Class 1 and Class 2. Both DES (40- and 56-bit) and RC5 encryption (64-, 128- and 168-bit) are supported.  See "WAP Browser configuration: Security" on page 13 for more information.	v.3.3
Web access restriction	✓			To restrict wireless web access, system administrators can turn the BlackBerry MDS Connection Service on or off for specific users or user groups. System administrators can also set specific policies to control which corporate servers each user can access and which servers can initiate push connections to the BlackBerry MDS Connection Service.	v.3.2
Triple DES encryption	✓			Browser content is encrypted between the BlackBerry Enterprise Server and the device using Triple DES encryption, the same encryption used when sending email messages.	v.3.2
Network authentication	✓		✓	<b>BlackBerry Browser:</b> The BlackBerry Browser configuration supports several types of network authentication, including Basic Authentication, NT LAN Manager (NTLM), and Kerberos™.  <b>WAP Browser:</b> The WAP Browser configuration supports the Password Authentication Protocol (PAP), which is used for authentication against Remote Authentication Dial In User Service (RADIUS) for Packet Data Protocol (PDP) context activation on GPRS networks. PDP context activation enables data transmission between the network and the device.	v.3.2
WAP Push applications			✓	A WAP Push service record must be provisioned on the device to push data to the device. WAP Push service records are usually sent to the device during registration.	v.3.2
<b>Usability</b>					
Single browser icon on the device Home screen	✓	✓	✓	Users specify the default browser configuration.  To use a specific browser configuration for a specific web site, users can change to the appropriate browser configuration from the Go To dialog box.	v.4.2
Bookmarks associated with browser configuration	✓	✓	✓	Bookmarked web sites are stored with an associated browser configuration. When a bookmark is opened, it automatically uses the associated browser configuration.	v.4.2
On-device help available	✓	✓	✓	A browser Help page is available from the browser menu.	v.3.7



## Browser content support

Content type	Description	Supported as of
<b>Markup languages (See “Determining which markup languages are accepted” on page 20 for more information.)</b>		
WAP CSS (partial support)	WAP CSS is defined by the WAP Forum. See WAP-239-WCSS-20011026-a at <a href="http://www.wapforum.org">http://www.wapforum.org</a> for more information.	v.3.8
XHTML Mobile Profile (XHTML-MP)	This subset of XHTML 1.1 is defined by the WAP Forum. See WAP-277-XHTMLMP-20011029-a at <a href="http://www.openmobilealliance.org">http://www.openmobilealliance.org</a> for more information.	v.3.6
HTML	HTML is defined in a W3C specification. The browser ignores tags that are not present in the XHTML-MP subset. Visit <a href="http://www.w3.org/TR/html401">http://www.w3.org/TR/html401</a> for more information about this specification.	v.3.2.1
Compact HTML (cHTML)	This subset of HTML 2.0, HTML 3.2, and HTML 4.0 is as described in a W3C Note (NOTE-compactHTML-19980209). Visit <a href="http://www.w3c.org">http://www.w3c.org</a> for more information.	v.3.2.1
WML 1.3	WML is defined by the WAP Forum. See WAP-191-WML-20000219-a at <a href="http://www.wapforum.org">http://www.wapforum.org</a> for more information.	v.3.2
<b>Images (See “Image conversion” on page 20 for more information about converting images.)</b>		
JPEG	Only color devices support .jpg images directly. With the BlackBerry Browser and Internet Browser configurations, the network gateway converts .jpg images for display on monochrome devices.	v.3.7
GIF	The device supports both the GIF87 and GIF89 image formats. The browser supports animated .gif files.	v.3.2.1
PNG	By default, the browser converts .gif images to .png images. The PNG format has a higher compression ratio and it supports alpha channels.  Java developers can write transcoders to convert other image formats to PNG format.	v.3.2.1
Wireless bitmap (WBMP)	The browser supports monochrome WBMP images. See the <i>Wireless Application Environment Defined Media Type Specification</i> (WAP-237-WAEMT-20010515-a) at <a href="http://www.wapforum.org">http://www.wapforum.org</a> for more information on wireless bitmaps.	v.3.2
<b>Complex content presentation formats</b>		
Framesets	The browser supports framesets and frames. The browser renders framesets by stacking frames vertically on a single page in the order in which the browser encounters them, using the full width of the device screen and as much vertical space as is required to contain all the frames. The browser processes one frame at a time, rendering the frame completely before processing the next frame.  Inline frames, specified using the <code>&lt;iframe&gt;</code> element, are not supported.	v.4.2
Web feeds	The browser supports the RSS0.9, RSS1.0, RSS2.0, and ATOM web feed formats.  The browser renders web feed content in a manner similar to the message list; it lists items by date, with unread items in bold. When the user opens an item to view more content, the content appears in a new page.  Users can add web feeds to the bookmark list.	v.4.2

Content type	Description	Supported as of
<b>Mobile media</b>		
Media Engine content	<p>The browser supports mobile media in PME and PMB formats. These formats are binary file formats that are optimized for wireless downloading to wireless devices and can be read by the Media Engine, a component of the BlackBerry Device Software.</p> <p>Content developers can author mobile media in several ways:</p> <ul style="list-style-type: none"> <li>using the Plazmic® Composer (available as part of the Plazmic Content Developer's Kit) to create .pme or .pmb files. Visit <a href="http://www.plazmic.com">http://www.plazmic.com</a> for more information.</li> <li>creating .svg files then converting them to .pme files using the Plazmic SVG Transcoding Utility.</li> </ul> <p><b>Note:</b> The network gateways for the BlackBerry Browser and Internet Browser configurations also include a transcoder that converts SVG content into PME format before sending the content to the device.</p> <p>To view mobile media content on a device, visit <a href="http://mobile.blackberry.com">http://mobile.blackberry.com</a>.</p>	v.3.7
Audio	<p>The browser allows you to download audio files of up to 128 KB in size. It supports the following audio MIME types:</p> <ul style="list-style-type: none"> <li>audio/adpcm</li> <li>audio/mid</li> <li>audio/midi</li> <li>audio/x-midi</li> <li>audio/x-oki-adpcm2</li> </ul>	v.3.7
<b>Scripts</b>		
JavaScript	<p>The browser supports JavaScript 1.3 and earlier, and subsets of JavaScript 1.4 and 1.5. The browser also supports the ECMA-262 ECMAScript Language Specification.</p> <p><b>Note:</b> JavaScript is supported only on BlackBerry devices that are 16MB or greater.</p> <p>The browser processes JavaScript that runs when the page is first rendered, and JavaScript that is associated with control actions on the page. The JavaScript support manages any additional HTML content and additional JavaScript content that the JavaScript produces, and reads any auxiliary JavaScript support libraries that are referenced from the page.</p> <p>System Administrators can turn on JavaScript support using IT policy. Users can turn on JavaScript support in the BlackBerry Browser Options, accessed from the browser menu.</p> <p><b>Note:</b> The browser does not support style sheets for Dynamic HTML. As a result, any JavaScript on the page that creates Dynamic HTML effects (for example, pop-up menus) runs but has no visual effect, and might not be fully functional. The browser supports JavaScript for pages whose HTML content is partially or fully generated by JavaScript, and for data processing operations coded in JavaScript, such as input field validation in forms and processing "Login" buttons on various sites.</p>	v.3.8
WML scripts	<p>The browser supports WML Script 1.2.1. WML Script lets you manipulate data values between WML decks programmatically. Common operations that are performed in WML Script include input validation, input aggregation, and conditional form processing.</p> <p>See <i>WML Script Specification</i> (WAP-193-WMLS-20001025-a) and <i>WML Script Standard Libraries Specification</i> (WAP-194-WMLSL-20000925-a) at <a href="http://www.wapforum.org">http://www.wapforum.org</a> for more information.</p>	v.3.2.1

Content type	Description	Supported as of
<b>Application pushes and downloads</b>		
Wireless application downloads	The browser supports the MIDP 2.0 wireless provisioning standard for wireless application downloads. When a user downloads an application to the device, the application installs automatically and the application icon appears on the Home screen.	v.3.6
BlackBerry MDS Connection Service reliable push	With BlackBerry MDS Connection Service reliable push, the browser sends confirmation messages from the device for successful application downloads and pushes.	v.3.8
Push applications	<p>Organizations can write push applications that send new web content and alerts to specific users automatically. With push applications, information can be delivered to the device as it becomes available; users do not have to request or download the data. For example, instead of relying on users to find intranet content, an organization can transmit data proactively. Users do not have to connect to corporate servers to check for new content; instead, an alert can be sent to users when new content is available.</p> <p>Both the BlackBerry and WAP Browser configurations support push applications, but they support them in different ways. See "Creating browser push applications" on page 67 for more information. The Internet Browser configuration does not support push applications.</p>	v.3.2
<b>Customization</b>		
APIs for third-party applications	<p>Two APIs assist with third-party integrations with the browser:</p> <ul style="list-style-type: none"> <li>• BrowserField API: Third-party applications can use the BrowserField API to embed an HTML, WML, or PME field anywhere within their application.</li> <li>• HTTP Filters API: The HTTP Filters API enables code on the device (usually third-party application code) to register itself with the browser as the provider of content from a specified URL. Third-party developers can then focus on content generation and application logic by creating code that uses the browser as its UI engine.</li> </ul>	v.3.8

## Managing multipart content

One of the factors that limits the speed with which the browser can download and render content is the prevalence of “composite” web pages (pages composed of a main WML or HTML page and one or more related auxiliary files, such as style sheets, JavaScript files, or image files). Because each auxiliary file requires that the browser submit a separate HTTP request, rendering times can be extended considerably.

To improve efficiency, content developers are increasingly posting all the necessary parts of a composite web page in a single bundle, letting the browser download all required content with a single request. The Content-Type header in the response identifies the content as a multipart bundle.

The browser supports the following Content-Type values:

- **multipart/mixed:** Denotes a collection of independent files. The first component of the bundle is assumed to be the principal file, for example, the WML or HTML page. Subsequent parts are assumed to be the auxiliary files, such as images, JavaScript files, or style sheets.
- **multipart/related:** Denotes compound documents, where the document is built from the pieces contained in the bundle. The browser cannot display the content without all the aggregate components.
- **multipart/alternative:** Denotes a collection of alternative versions of the same content. The browser searches until it finds a component with content that it can render.

## Determining which markup languages are accepted

Service providers and system administrators can specify whether the browser accepts WML or HTML content, or both, based on the type of content, the browser configuration, and the gateway behavior. Device users can change this value later. For example, to prevent unwanted content conversions, system administrators might want the BlackBerry Browser and Internet Browser configurations to indicate support for both HTML and WML. In other cases, service providers might want the WAP Browser configuration to indicate preference for WML content so that web sites with both WML and HTML content send the WML version and minimize bandwidth usage.

In “HTML only” mode, if a requested URL returns WML content, the web server or gateway returns an HTTP response of 406 (“Not Acceptable”). The browser then adds the WML capability to the HTTP\_ACCEPT header and requests the URL again.

## Image conversion



**Note:** The image conversion feature requires one of:

- BlackBerry Enterprise Server for Microsoft® Exchange Version 3.6 or later
- BlackBerry Enterprise Server for IBM® Lotus® Domino® Version 2.2 or later
- BlackBerry Enterprise Server for Novell® GroupWise® Version 4.0 or later
- BlackBerry Internet Service Browsing

To provide the information that the gateway needs to convert images so that they can be rendered by the browser, the browser includes the following HTTP headers with every request:

- **Transcode Content (X-RIM-transcode-content):** This header enables image optimization processing by indicating which image types should be converted.

- **User Agent Profile (Profile):** This header contains a URL that includes the following information:
  - location of the User Agent Profile (UAProf) documents on the Internet
  - device model number
  - BlackBerry Device Software version

See the *User Agent Profiling Specification* (WAP-248-UAProf-20011020-a) at <http://www.wapforum.org> for more information.

## Image conversion for the BlackBerry Browser and Internet Browser configurations

For both the BlackBerry Browser and Internet Browser configurations, back-end services read the information in the User Agent Profile document to determine device capabilities, such as screen size, color depth, and accepted image and content types (for example, only color devices can display .jpg images directly).

With this information, the network gateway can return device-appropriate content to the browser. The HTTPcontenttranscoderslist.property file stores image and content conversion information. By default, the BlackBerry MDS Data Optimization Service converts JPEG format images into PNG format for display on monochrome devices. The BlackBerry MDS Data Optimization Service converts images in BMP and GIF formats to PNG format for all devices.

## Image conversion by the WAP gateway

With the WAP Browser configuration, images are typically not converted. Most WAP gateways pass along all images to the browser, provided the ACCEPT header lists the appropriate image format and the image is within the allowable size; however, some WAP gateways might restrict the image types or impose size restrictions that prevent larger image files from being retrieved.

## Image processing

The browser loads images differently depending on the gateway that is used.

Gateway	Image processing
<ul style="list-style-type: none"> <li>• BlackBerry Enterprise Server for Microsoft Exchange Version 3.6 or later</li> <li>• BlackBerry Enterprise Server for IBM Lotus Domino Version 2.2 or later</li> <li>• BlackBerry Enterprise Server for Novell Groupwise Version 4.0 or later</li> <li>• BlackBerry Internet Service</li> </ul>	The network gateway retrieves images while it processes the HTML content, and includes the images when it sends the HTML or XHTML content to the device. Images appear immediately.
<ul style="list-style-type: none"> <li>• WAP gateway</li> <li>• BlackBerry Enterprise Server for Microsoft Exchange Version 3.5</li> </ul>	The browser first displays image placeholders, with the alternate text of each image (if alternate text is provided). In the background, the browser loads each image separately and updates the page as each image becomes available. The browser retrieves images from the local cache, if possible.

When processing images for display on the device, the following techniques apply:

- **Horizontal scaling:** Images that exceed the screen dimensions are scaled to be no wider than the screen content area. To maintain the aspect ratio, images are scaled by the same factor in both the horizontal and

vertical dimensions. The screen content area is the screen width minus 5 pixels. The vertical scroll bar that displays at the right is 5 pixels wide. Users can press the Alt key and roll the trackwheel to scroll horizontally.

- **Vertical scaling:** Images that are more than twice the screen height are scaled to twice the screen height. To maintain the aspect ratio, images are scaled by the same factor in the horizontal and vertical dimensions. Users can roll the trackwheel to scroll vertically.
- **Size limitation:** If a monochrome image is more than 8192 bytes after scaling, the image is discarded. If the image is part of a link, the browser displays the alternate text for the image.
- **Color:** On a monochrome device, color images are dithered into a monochrome image. On a color device, the image color depth is reduced to accommodate the number of colors that the device supports.
- **Vertical alignment:** The vertical alignment in `<img>` tags is ignored.

The network gateway for the BlackBerry and Internet Browser configurations scale and dither images for the BlackBerry device. With the WAP Browser configuration, the device performs scaling and dithering as necessary.



**Note:** In BlackBerry Device Software Version 3.6 or earlier, the browser displays images in a vertical area that is separate from the content before and after the image. Images do not appear inline with surrounding text; they are aligned horizontally according to the ALIGN attribute (ALIGN=LEFT or ALIGN=RIGHT).

# Browser interface and features

Browser screen  
Browser features

## Browser screen

By default, the browser is designed to display a non-scrolling title bar at the top of each page that displays the following items:

- page title
- unread messages
- pending service books
- connection information
- security settings
- network signal strength

When the browser requests or loads pages, a progress bar appears at the bottom of the screen.

On WML pages, <do> elements appear both as soft keys in the non-scrolling section at the bottom of the page and as menu items on the browser menu. See “Browser menus” on page 23 for more information.

Links to web pages, phone numbers, and email addresses are underlined with a dotted line. See “Links” on page 24 for more information.

When a web page does not fit on one screen, a vertical scroll bar appears on the right side of the screen.

## Browser menus

The browser menu provides access to most tasks that users perform when they browse. The menu provides the standard BlackBerry system menu items, such as **Hide Menu** and **Close**, a **Help** item, and appropriate context menu items, such as **Select** and **Find**.

The browser provides standard menu items for navigation, including **Home**, **Back**, **Forward**, **History**, and **Refresh**. The **Go To** menu item lets users type any URL.

Specific menu items appear depending on the page and the item that is selected.

## WML <do> elements

The browser displays WML <do> elements in the following ways:

- as soft keys in the non-scrolling area at the bottom of the screen. A maximum of two soft keys can be displayed on the screen.
- as items on the browser menu. All <do> elements included in the current WML card are displayed in the browser menu.

Soft keys provide a familiar way to navigate pages for users who have used WAP browsers on other mobile devices, such as cellular phones. To select soft keys, users scroll to the soft key and click the trackwheel.

## Links

Links are underlined with a dotted line. In BlackBerry Device Software Version 3.7 or later, the browser provides one-click navigation. To follow a selected link, users click and hold the trackwheel or trackball, or press the Enter key.

Link Type	Description
Web page links	On a web page, users scroll to links by rolling the trackwheel or trackball. Scrolling up or down moves to the next or previous link on the same line, before moving to the next line.
Image links	When images are placed in <a> or <anchor> tags, users can select the image to follow the link or to perform the associated action. If the size of the image exceeds the screen dimensions, users can click the <b>Full Image</b> menu item to load the image on a new page. When the browser encounters a full image request, the image is retrieved in its original, unscaled form and is transmitted to the browser. The browser renders the image from the top left. Vertical and horizontal scroll bars are available. To scroll vertically, users roll the trackwheel or trackball. To scroll horizontally, users press the <b>Alt</b> key and rolls the trackwheel or trackball.
Image maps	The browser supports image maps. When images contain the image <map> tag, users can select links on portions of an image. A dotted line around the hot spot denotes a link. To select the link, users click and hold the trackwheel or trackball, or press the Enter key.
Phone links	<p>The browser supports the following types of phone links:</p> <ul style="list-style-type: none"> <li>• WTAI) Make Call links (URI form): &lt;a href="wtai://wp/mc;14165551212"&gt;Call office&lt;/a&gt;</li> <li>• phone links in i-mode format: &lt;a href="tel:14165551212"&gt;Call office&lt;/a&gt;</li> <li>• Direct Connect links on iDEN® networks: &lt;a href="dc:234*234*234"&gt;Call office&lt;/a&gt;</li> <li>• Computer Telephone Integration (CTI): &lt;a href="cti:333333"&gt;Call office&lt;/a&gt;</li> </ul> <p>When users click a phone link, the phone opens and a dialog box appears. Users select whether to make the call or not.</p>
Email links	<p>The browser supports mailto: links. For example:</p> <pre>&lt;a href="mailto:kate.turner@blackberry.com"&gt;Email Kate&lt;/a&gt;</pre> <p>When users click an email link, the Compose screen appears with a new message.</p>



## Option lists

Option lists are displayed as radio buttons (for single-selection lists) or check boxes (for multiple-selection lists). To select an option in a list, users press the **Space** key or click the **Select Option** menu item.

In WML, if a single-selection list does not have an `onpick` action defined for one of its options, when the user selects an option, the browser runs either the first `<do>` action with a `type` of `accept`, or, if none has the `type` `accept`, the first `<do>` action listed.

In HTML and XHTML, options that are grouped in a `SELECT` tag appear in a drop-down list. To select an option, users click the drop-down menu and click **Change Option**.

If a `size` is defined, the selection list appears in a vertical list instead of a drop-down list.

## Browser features

### History

The browser maintains a navigation history of up to 20 items. When a user navigates to a page, the browser adds the URL of that page to the navigation history.

When the history list reaches 20 items, the browser replaces the earliest pages with the new URLs. If memory on the device becomes low, the browser removes history items to free memory.

When the user navigates to a previous page and selects a new link from that page, the browser removes any URLs after that point in the history. The URL of the newly selected page becomes the next item in the history sequence. For example, if a user navigates from a music page to a news page, the history displays both pages. If the user then navigates back to the music page and selects a genre of music, the history displays the music page and the genre page; the news page is no longer listed.



**Note:** If a user visits a WML page by some manner other than a predefined link (for example, not a bookmark or the Go To dialog box), or if that page has a `newcontext` attribute defined, the browser automatically clears the history before displaying the new URL. This behavior is required to conform to WML security specifications.

### Cookies

The browser stores cookies on its own behalf. The browser supports standard cookies based on the RFC 2109 *HTTP State Management Mechanism*, as well as the Netscape® format for expiry dates (`EXPIRES=Weekday, DD-Month-YY HH:MM:SS GMT`). The browser maintains cookies when the device is turned off.

If configured to do so, the BlackBerry MDS Connection Service can store cookies on behalf of the BlackBerry Browser. This service supports cookies based on the RFC 2965 *HTTP State Management Mechanism*, which supports the HTTP state management header `Set-Cookie-2`. See the *BlackBerry Enterprise Server Administration Guide* for more information.

## Cache

The browser maintains three caches—a content cache, a channel cache, and a cookie cache—each of which stores a different type of information.

Cache	Description
Content	This cache includes the raw data cache; it contains all data that is cached as a result of normal browser activity.
Channel	This cache contains data that is sent to the device by a channel or cache push.
Cookie	This cache contains cookies that are assigned to the browser by visited web pages.

Whenever possible, the browser loads requested content from the local cache. Users can clear the content, channel, and cookie caches on the device to free remaining memory and refresh any visited web pages.

The browser respects cache control directives that web servers send in response headers, such as `Expires`, `Max-Age`, and `Cache-Control`. When permitted, the browser retrieves content from the cache based on associated cache control directives. See the specification WAP-120-UACACH-20010413-a at <http://www.wapforum.org> for more information on WAP user agent caching.

On the BlackBerry device, the browser saves the channel and cookie caches in persistent storage, so information is saved even if the device is reset. The browser clears an item from the cache when it expires. If an item has no expiry time explicitly set, the browser clears the item from the cache after 29 days.

On devices with 8 MB of memory, the browser clears the content cache when the user closes the browser session. On devices with 16 MB of memory, the content cache persists. The device clears items from the cache to free memory when necessary, with expired pages cleared first.

Service providers can set the size of the raw data cache. The system default sizes of the raw data cache for the various BlackBerry devices are:

- 200 KB for devices with 8 MB of memory
- 500 KB for devices with 16 MB of memory
- 2 MB for devices with more than 16 MB of memory

Users cannot view or change these options.



**Note:** The channel and cookie caches are stored persistently; however, information in the content cache is lost if the device is reset or the BlackBerry MDS Connection Service is restarted.

## Bookmarks

The browser provides bookmark support that combines the functionality that is typical of computer-based browsers with added features that are designed specifically for the wireless environment.

As with desktop browsers, users can add bookmarks for any web site that they visit, and they can create a hierarchy of folders to organize them. Each browser configuration has its own set of bookmark folders so that users can organize bookmarks separately, based on which browser configuration is best for the bookmarked site. Bookmark folders for all browser configurations can be accessed regardless of the browser configuration that the user is browsing with. Users can move or copy bookmarks between any folder. They can edit the title and URL of bookmarks as necessary, and they can search for and delete specified bookmarks.

During the registration process, service providers can insert a set of customized bookmarks in the browser. These bookmarks are sent over the wireless network in the browser configuration service record. By default, the bookmarks are placed in the Bookmarks tree view hierarchy in a new folder called Carrier Bookmarks. The name of the folder can also be customized.

Frequently used browser pages can also be saved to the Messages screen for quick access.

Bookmarks are useful when users are outside a wireless coverage area. When users add a bookmark, they can make the bookmark available offline, which means that both the content and URL of the page are saved. Offline bookmarks are maintained even if the device is reset.

Offline bookmarks for web pages that contain forms also save the current values of form fields, which users can use as a template for frequently used forms. For example, users can add an offline bookmark for a page that contains a form. Later, even if users are outside a wireless coverage area, they can load the bookmarked form, fill out the appropriate fields, and submit the form. Users can then save the browser request to the message list. When the device returns to a wireless coverage area, the browser sends the request automatically.

Users can back up browser bookmarks using the BlackBerry Desktop Software, so that when they update their devices with new applications, their bookmarks are retained.



# Designing wireless web content for the BlackBerry Browser

Creating effective content for the BlackBerry Browser  
Creating effective images  
Defining queues for offline form submission  
Making requests for content only when content has changed  
Delivering device-specific content

## Creating effective content for the BlackBerry Browser

Many factors (such as the network gateway, the browser configuration, and the BlackBerry device memory, screen size, and color depth) influence how content renders on the device.

### Follow basic web design principles

Many standard principles of web site design apply when you create content for wireless devices. Consider the following design recommendations when you plan your web site:

- **Understand your audience:** Determine who will use the site and the primary service that your site will provide.
- **Create an appropriate site hierarchy:** Structure your web site based on its purpose, and organize the site to minimize the time that it takes users to find information or perform tasks.
- **Provide useful links:** Minimize the number of pages that users must navigate to accomplish their goals and consider the following guidelines for links:
  - Include a link to the home page on each page.
  - Whenever possible, include links to other related pages on your site, to minimize backward navigation using the browser history.

### Organize content effectively

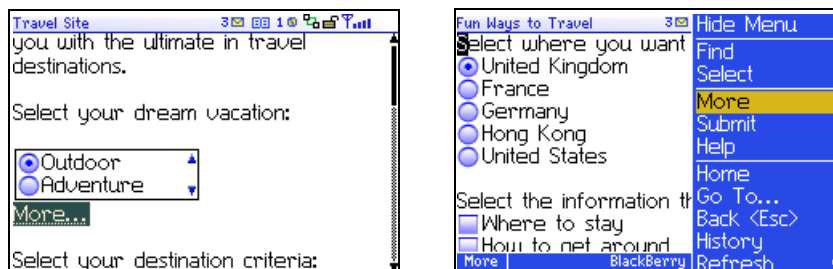
Consider the following guidelines when planning your web site:

- **Deliver related content on as few pages as possible:** Although a page with more content might take a few seconds longer to download, users do not have to make subsequent requests, and the information is available

even when users move outside a wireless coverage area. BlackBerry users can use the trackwheel to scroll through several screens of text easily.

On WML pages, put related cards in the same deck whenever possible so that the document has to be loaded only once. If the deck contains a relatively large card that many users might not want to view, save the card in its own deck to minimize download time.

- **Add links to related content:** If you divide related content into more than one page, make links to related content easily accessible. Make sure that links to related content are visible in a non-scrolling area of the page or at the top of the page. For example, in WML, you could add a **More** menu item (or soft key) to let users retrieve related content quickly.



Example of a More link and menu item

## Select the most appropriate markup language

When you create a new web site, you must decide whether you are going to write the source in HTML, WML, or SVG. Consider the following advantages and disadvantages of each markup language as you decide.

Markup	HTML/XHTML	WML	SVG
<b>Advantages</b>	<ul style="list-style-type: none"> <li>• HTML can be migrated to XHTML much more easily than to WML</li> <li>• XHTML supports greater layout versatility than WML</li> <li>• functionality can be extended considerably using JavaScript</li> <li>• as of WAP 2.0, XHTML-MP has become the markup language supported by the Open Mobile Alliance and will become the standard for mobile devices</li> </ul>	<ul style="list-style-type: none"> <li>• most users of wireless web sites are accustomed to WML</li> <li>• currently the most widely used markup language for wireless web applications</li> <li>• has a well-maintained DTD and is well-documented</li> <li>• functionality can be extended using WMLScript</li> </ul>	<ul style="list-style-type: none"> <li>• lets content developers add movement and sound to their content</li> <li>• offers dynamic layout and presentation support</li> <li>• automatically transcoded to the .pme file format by the BlackBerry MDS Data Optimization Service or the BlackBerry Internet Service</li> </ul>
<b>Disadvantages</b>	<ul style="list-style-type: none"> <li>• usually larger than WML content, so it can take longer to display</li> </ul>	<ul style="list-style-type: none"> <li>• supports only basic page layout; best suited to very basic sites</li> <li>• WMLScript is much less robust than JavaScript</li> <li>• WML has been deprecated by the Open Mobile Alliance as of WAP 2.0 in favor of XHTML-MP</li> </ul>	<ul style="list-style-type: none"> <li>• SVG is not supported by the browser directly; it must be transcoded to the .pme file format (either by a data optimization service provided with the network gateway, or by the content developer)</li> <li>• takes longer to download than other formats</li> </ul>

## Consider BlackBerry device screen sizes

Design web pages to use the BlackBerry device screen effectively. BlackBerry devices have larger screens than many other mobile devices, such as mobile phones. Depending on the device type and selected font size, the browser can typically display 12 to 18 lines of text with 28 to 35 characters on each line. In contrast, many mobile phone browsers display 4 to 7 lines of text, with 10 to 15 characters on each line.

## Encourage text entry

BlackBerry users can use the keyboard to type text into web forms.

The browser supports both `<input type="text">` and `<textarea>` elements in HTML, and `<input type="text">` in WML. With QWERTY or SureType™ keypads, text entry is considerably easier for BlackBerry users than for users of mobile phones, and they need not be avoided.

## Minimize download time

Download time is affected by three factors: content size, the wireless network, and protocol characteristics. For example, a 15-KB file can take 30 seconds or more to download through a WAP gateway on a GPRS network.

You can improve download time by reducing the size of web pages. To reduce the size of web pages, avoid unnecessary content and images. Reduce image file sizes as much as possible.

## Improve rendering time

Rendering on the browser does not affect the time it takes to display content as much as the download time does, but large content can still require several seconds to parse and display.

The BlackBerry MDS Services and the BlackBerry Internet Service will speed up rendering times by processing HTML content before sending it to the browser. These components filter out unsupported elements and convert content into a tokenized format that the browser can display efficiently.

## Creating effective images

Consider the following guidelines when you include images on your pages:

- Fonts that are saved as images should not be anti-aliased. Anti-aliasing smooths edges by blending the background and foreground colors. Anti-aliased images do not display optimally on the BlackBerry devices.
- If you resize an image to better fit the smaller screen, when possible, redraw the image. Scaling down the image results in blurred edges that display poorly.
- Although the network gateway for the BlackBerry Browser and Internet Browser configurations can dither color images to monochrome, ideally your images should be saved in monochrome format for display on monochrome devices. The following example demonstrates examples of the same page rendered on a color device and a monochrome device. See "Delivering device-specific content" on page 35 for more information about delivering device-appropriate content.



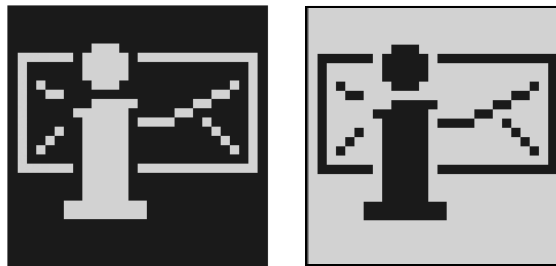
Effect of dithering on color images

- If it is not possible to provide both a color image and a monochrome image, verify that the image displays acceptably on both device types.
- Users can specify whether images are loaded or not; therefore, images should not be critical to the effectiveness and usefulness of your web site.

See “Browser content support” on page 17 for more information about supported image types and image processing.

## Create effective monochrome images

The following examples demonstrate monochrome images that display well on the BlackBerry device, and images that display poorly. The first pair of images display well because they are monochrome and contain well-defined edges. The second pair of images display poorly because of feathered edges and blurred colors.



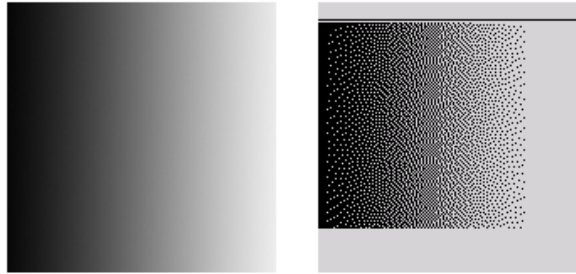
Examples of monochrome images that display well in the browser



Examples of images that display poorly in the browser



To convert an image to monochrome using Adobe® Photoshop®, convert your image to a bitmap using the 50 percent threshold method. You might need to discard any color information by converting the image to grayscale. The following diagram demonstrates how a gradient appears on the BlackBerry monochrome devices. Gradients appear adequately on the BlackBerry device screen, but they are less effective on a smaller scale. For example, a font with feathered edges appears poorly on the device screen.



Examples of grayscale gradients on a BlackBerry monochrome device

## Defining queues for offline form submission

If you define form-submission queues, BlackBerry users can complete and submit forms and continue browsing without waiting for the form to be submitted or worrying about whether they are in a wireless coverage area. Users can load an HTML form (or a WML page with inputs) in the browser, fill in the values, and then submit the form to an Offline Queues list. The browser continuously processes any queued forms and submits the forms in the background.

If the device is outside a wireless coverage area, users can still complete and submit several forms (possibly for different queues). The browser queues the form requests and submits them when the device is back in coverage.

After forms are submitted, user responses are stored by the browser. Users can open the queue list, and click a request to view the response.

The following HTTP headers allow you to create a form queue:

Parameter	Required?	Description
x-rim-queue-id	Yes	Specifies the Offline Form Queue to which any GET or POST requests from form submissions on this page should go. The value may be any text string.
x-rim-next-target	No	Specifies the next page to load after sending any GET or POST requests resulting from this page to the Offline Form Queue. The value may be any valid URL.
x-rim-request-title	No	Specifies the label used to identify this request in the Queue view page. The value may be any text string. By default, the request is identified using the title of the page.
x-rim-request-id	No	Specifies whether the browser will generate a unique ID and add it as an HTTP header for every offline request resulting from this page. The value may be a Boolean True or False. By default, this value is True.
x-rim-request-date	No	Specifies whether the browser will generate a time stamp and add it as an HTTP header for every offline request resulting from this page. The value may be a Boolean True or False. By default, this value is True.

You can create form queues using these headers either by creating an HTTP property file or by adding the queuing parameters directly to the HTML or WML page.

## Create an HTTP header property file

Creating an HTTP property file in which you define the queuing parameters lets you create and manage multiple form queues in a single location; however, it requires that you properly set your web server to send the headers when the web page containing the form is requested.

To create a queue for a form on `stock-monitor.xhtml`, for example, you might define the queuing parameters as follows:

```
<Files stock-monitor.xhtml>
  Header set cache-control max-age=2592000
  Header set x-rim-queue-id Register
  Header set x-rim-request-title "Stock Monitor"
  Header set x-rim-next-target success.xhtml
</Files>
```

You can add queuing parameters for additional forms within the same header file.

See "Offline form queue header file: .htaccess" on page 34 for a complete sample of the property file.

---

### Example: Offline form queue header file: .htaccess

```
<Files stock-monitor.xhtml>
  Header set cache-control max-age=2592000
  Header set x-rim-queue-id Register
  Header set x-rim-request-title "Stock Monitor"
  Header set x-rim-next-target success.xhtml
</Files>
<Files stock_monitor.wml>
  Header set cache-control max-age=2592000
  Header set x-rim-queue-id Register
  Header set x-rim-request-title "Stock Monitor"
  Header set x-rim-next-target success.wml
</Files>
<Files success.xhtml>
  Header set cache-control max-age=2592000
</Files>
<Files success.wml>
  Header set cache-control max-age=2592000
</Files>
```

---

## Add queuing parameters directly to the web page

Queuing parameters are managed differently by HTML pages and WML pages.

- In HTML or XHTML, queuing parameters are added using hidden `<input>` elements:

```
<input type="hidden" name="x-rim-queue-id" value="Register" />
<input type="hidden" name="x-rim-request-title" value="Stock Monitor" />
```

```
<input type="hidden" name="x-rim-next-target" value="success.xhtml" />
```

To see a sample XHTML page that defines queuing parameters, see "Creating an XHTML-MP page" on page 43.

- In WML, queuing parameters are added using `<postfield>` elements:

```
<input type="hidden" name="x-rim-queue-id" value="Register" />
<input type="hidden" name="x-rim-request-title" value="Stock Monitor" />
<input type="hidden" name="x-rim-next-target" value="success.wml" />
```

To see a sample XHTML page that defines queuing parameters, see "Creating a WML page" on page 56.

## Making requests for content only when content has changed

Download times in a wireless environment are typically slower than on a desktop browser. Therefore, downloading content that has not changed since it was previously downloaded to the device—and that is therefore already present in the browser cache—is both unnecessary and a waste of time.

By making GET requests conditional based upon whether the requested content is new, you can force the browser to download content only when the cached content is out of date.

To make a GET request conditional, content developers must use the following HTTP headers.

Header	Description
If-Modified-Since	<p>This header is used with a GET method to request that the web server transfer content only if it has been modified since the specified date and time. For example:</p> <pre>If-Modified-Since: Fri, 6 May 2005 12:00:00 GMT</pre> <p>The request is managed as follows:</p> <ul style="list-style-type: none"> <li>• If the requested content has been modified since the specified date, or if the specified date is invalid (such as a date that is later than the server's current time), the content will be downloaded as normal.</li> <li>• If the requested content has not been modified, the server sends a 304 (not Modified) response, and the browser retrieves the content from the cache.</li> </ul>
Last-Modified	<p>This header identifies the date and time at which the web content was most recently modified. For example:</p> <pre>Last-Modified: Fri, 6 May 2005 12:00:00 GMT</pre> <p><b>Note:</b> The exact meaning of this header field depends on the implementation of the origin server and the nature of the original resource:</p> <ul style="list-style-type: none"> <li>• For files, it may be just the file system last-modified time.</li> <li>• For entities with dynamically included parts, it may be the last-modified time of the most recently modified component.</li> <li>• For database gateways, it may be the last-update time stamp of the record.</li> </ul>

## Delivering device-specific content

When submitting an HTTP request, most desktop browsers, such as Microsoft® Internet Explorer and Netscape Navigator®, include a header that identifies the browser and version. The BlackBerry browser includes the `Profile` (User Agent Profile) header, which specifies the BlackBerry model and capabilities.

This header is a URL that uses the following form:

```
http://www.blackberry.net/go/mobile/profiles/uaprof/<BlackBerry-model>/
<software-version>.rdf.
```

where:

- *<BlackBerry-model>* is the BlackBerry device model number, for example, 6210 or 7210. The model number lets you determine the screen dimensions and the color depth; BlackBerry devices that have a model number that begins with the number six have a monochrome display. Devices that have a model number that begins with the number seven or eight have a color display.
- *<software-version>* is the BlackBerry Device Software version, for example, 4.0.0 or 4.1.0.

The information contained in this header lets you determine the content that is most appropriate for display on the browser making the request. You can create a script to extract this information, letting the content server return device-appropriate content in the HTTP response.



**Note:** There is no way to differentiate between HTML or XHTML pages that are designed for desktop browsers and pages that are designed for wireless devices. The MIME type `text/html` is used in both cases.

Other headers are also included, which help determine what content is sent. For example, the `Accept` header specifies the content types that the browser accepts. The preferred content type is determined by the order of content types in the `Accept` header, from left to right. For example, the following header indicates that WML is preferred over HTML, and .gif images are preferred over .png images:

```
Accept: text/vnd.wap.wml, text/html, image/gif, image/png
```

## Deliver device-specific content

1. Create device- and browser-specific content and images. For design tips, see "Creating effective content for the BlackBerry Browser" on page 29.
2. Copy your content files into browser-specific directories on your web application server.
3. Write a browser detection script that parses the Profile header to determine the browser type and the supported content types, and returns content that is appropriate for the requesting browser. See "Write a browser detection script" on page 36 for more information.
4. Copy the browser detection script to your web application server.
5. Test the script by requesting content from a variety of browsers and devices. You can use the BlackBerry Device simulator to simulate a variety of BlackBerry devices.

## Write a browser detection script

Write a browser detection script using any scripting language that lets you access and manipulate HTTP headers. The following example demonstrates how to write a browser detection script using Perl. The complete code sample is provided at the end of this section. See "Browser detection script (index.pl)" on page 37 for more information

1. Assign variables for the `Accept` and `User Agent Profile` headers.

```
$content = $ENV{'HTTP_ACCEPT'};
$browser = $ENV{'HTTP_PROFILE'};;;
```

2. Determine whether the browser accepts HTML content. If it does, parse the User-Agent field to determine whether this is a BlackBerry browser or a standard browser.

```

if ($content =~ html) {
    if ($browser =~ BlackBerry) {
        print "Location: http://mobile.blackberry.com/index.html", "\n\n";
    }
    elsif ($browser =~ Mozilla) {
        print "Location: http://www.blackberry.com/index.shtml", "\n\n";
    }
}

```

3. If the browser does not accept HTML, determine whether the client accepts WML content. If it does, parse the User-Agent field to determine whether this is a BlackBerry browser or another type of device.

```

elseif ($content =~ wml) {
    if ($browser =~ BlackBerry) {
        print "Location: http://mobile.blackberry.com/wml/index.wml", "\n\n";
    }
    else {
        print "Location: http://www.blackberry.com/unsupported.html", "\n\n";
    }
}

```

4. Provide a default web page for all other browser types.

```

else
{
    print "Location: = http://www.blackberry.com/index.html", "\n\n";
}

```

---

#### Example: Browser detection script (index.pl)

```

# Copyright (C) 2004 Research In Motion Limited.
# Note: URLs are used in this example for non-existent web sites.
#!c:\perl\bin\

$content = $ENV{'HTTP_ACCEPT'};
$browser = $ENV{'HTTP_USER_AGENT'};

if ($content =~ html) {
    if ($browser =~ BlackBerry) {
        print "Location: http://mobile.blackberry.com/index.html", "\n\n";
    }
    elsif ($browser =~ Mozilla) {
        print "Location: http://www.blackberry.com/index.shtml", "\n\n";
    }
}
elseif ($content =~ wml) {
    if ($browser =~ BlackBerry) {
        print "Location: http://mobile.blackberry.com/wml/index.wml", "\n\n";
    }
    else {
        print "Location: http://www.blackberry.com/unsupported.html", "\n\n";
    }
}

```

```
else {
    print "Location: http://www.blackberry.com/index.shtml", "\n\n";
}
```

---

## Send device-appropriate images

This section describes how to write a Microsoft® Visual Basic® script that parses the `Profile` header and returns device-appropriate images. The complete code sample is provided at the end of this section.

1. Create an Active Server Page.
2. At the top of your content file, specify the script language.

```
<%@ Language=VBScript %>
```

3. Specify the content type.

```
<%
'send the right MIME type
Response.ContentType = "text/vnd.wap.wml"
%>
```

4. Include the document declaration and markup language tags.
5. Parse the `Profile` header and determine the BlackBerry device model number. You can use the BlackBerry model number to return a color or monochrome image.

```
<p><%
' Detect colour devices (model number 7XXX)
uaprof = request.servervariables("HTTP_PROFILE")
```

6. Specify the format of the `Profile` header.

```
' Format is:
' http://www.blackberry.com/go/mobile/profiles/uaprof/<model-number>/...
```

7. To accommodate monochrome devices by default, specify the BlackBerry 6210 Wireless Handheld™ as the default model. By default, the server includes monochrome images in the HTTP response.

```
model="6210"
```

8. Include an `If` statement to verify that a BlackBerry device sent the `Profile` header.

```
If ( InStr( uaprof, "http://www.blackberry.com/go/mobile/profiles/uaprof/" ) = 1 )
```

9. Assign the 4-character string starting at position 53 to the model variable.

```
Then
    model = Mid( uaprof, 53, 4 )
End If
```

10. Include an `If` statement to return a color image if the device supports color, and a monochrome image if it supports black and white images only.

```
if ( model >= 7000 ) Then
    Response.write( "<img src=""images/BBLogoClr.gif"" alt=""BlackBerry""/><br/><br/>" )
Else
    Response.write( "<img src=""images/BBlogo.wbmp"" alt=""BlackBerry"" />&nbsp;" )
```

```
End If
%>
```

## 11. Include the rest of the page content.



**Tip:** To obtain more BlackBerry device information, perform the following steps:

1. Retrieve the .rdf document that the Profile URL contains.
2. Parse the XML content.
3. Search for an entity in the XML document, for example, "prf:ColorCapable".

---

### Example: Color device detection

```
# Copyright (C) 2004 Research In Motion Limited.
# Note: URLs are used in this example for non-existent web sites.

<%@ Language=VBScript %>
<%
'send the right MIME type
Response.ContentType = "text/vnd.wap.wml"
%

<p align="center">

<%
' Detect colour devices (model number 7XXX)
uaprof = request.servervariables("HTTP_PROFILE")

' Format is:
' http://www.blackberry.com/go/mobile/profiles/uaprof/<model-number>/...
' Set the default model to be the 6210
model="6210"
If ( InStr( uaprof, "http://www.blackberry.com/go/mobile/profiles/uaprof/" ) = 1 )
Then
    model = Mid( uaprof, 53, 4 )
End If

if ( model >= 7000 ) Then
    Response.write( "<img src=""images/BBLogoClr.gif"" alt=""BlackBerry""/><br/>
<br/>" )
Else
    Response.write( "<img src=""images/BBlogo.wbmp"" alt=""BlackBerry"" />&nbsp;" )
End If

%>
```

---





# Creating XHTML pages

Using XHTML-MP

Creating an XHTML-MP page

Code sample: Creating an XHTML-MP web page

## Using XHTML-MP

XHTML-MP is a subset of XHTML 1.1, extended with some WAP-specific tags. It has been designed for use by devices with smaller displays and limited memory and CPU power. XHTML 1.1 is a superset of HTML 4.0. See WAP-277-XHTMLMP-20011029-a at <http://www.openmobilealliance.org/tech/affiliates/wap> for more information.

The browser ignores any tags that are not part of the XHTML-MP subset and displays content as if the enclosing tags are not present. See “XHTML-MP reference” on page 101 for more information.

## Creating XHTML-MP–compliant sites

See “Browser content support” on page 17 for more information about browser content and image support.

- **Syntax:** Although it uses many standard HTML tags, XHTML-MP follows XML syntax conventions. This means that, unlike HTML, XHTML-MP content must be well-formed.
  - Every opening tag—including standalone tags such as `<p>`, `<br>`, and `<img>`—must either have a corresponding closing tag or be self closing.
  - Elements and attributes must be lowercase.
  - Attributes must be in quotations.
  - Elements must be properly nested so that no element overlaps another element.
- **Styles and style sheets:** The browser supports both internal and external style sheets, as well as the `STYLE` attribute that is used to provide an inline style for specific elements. The `<link>` tag is used to include external style sheets, while the `<style>` tag encloses internal element styles.

The browser supports a limited subset of commonly used properties of the WAP CSS. See “WAP CSS reference” on page 112 for more information about supported CSS properties.

The browser supports both background colors and background images.

- **Fonts:** Note the following font limitations:
  - Many web clients cannot display fonts other than monospace fonts.
  - The browser supports bold and italic fonts, as well as fixed-width fonts. Text that is enclosed in display tags, such as `<strong>` and `<em>`, appears in bold or italic, respectively. Other content-based style tags, such as `<code>`, `<samp>`, and `<kbd>`, are displayed in a fixed-width font.

- Text placed at angles and other text extension elements are not supported. To create text object effects, consider using SVG for your mobile media content.
- **Scripts and events:** The browser supports JavaScript, but it does not support applets. The browser reads the contents of a `<script>` tag, including any event-handler attributes. Only a limited subset of event handlers are supported. See "JavaScript language reference" on page 133 for more information.
- **Forms:** XHTML-MP supports basic forms. The standard keyboard on BlackBerry devices lets users easily type text into forms. File and image input types are not supported. Developers who want to create forms for writing content to the device file system should consider writing a Java application.

The browser allows you to set up queues so that users can fill out and submit forms even when they are outside of a wireless coverage area. See "Creating an XHTML-MP page" on page 43 for more information.

- **Tables:** The browser supports tables. Users can turn on HTML table support to properly display tables that fit within the screen width of the device. Larger tables are split apart with the table cells displayed in vertical sequence. Tables cannot be nested within each other.

You should test tables thoroughly in the BlackBerry Browser to verify that they display appropriately on the device.

- **Frames:** The browser supports the `<frameset>` and `<frame>` elements, but does not support inline frames (the `<iframe>` element). Because of the difficulty in rendering the complex page layouts that frames provide on smaller BlackBerry device screens, the layout specified for the frameset using the `<cols>` and `<rows>` elements are ignored. Instead, frames are rendered vertically in a single column.

If a web page contains a frameset, the browser displays the content of each individual frame vertically in the order in which they are encountered. The browser processes one frame at a time, and it renders each frame completely before processing the next frame.



**Note:** Support for frames is intended to improve usability when accessing content designed for desktop browsers; you should not include frames when designing content specifically for BlackBerry devices.

The browser also supports the following complex frameset behaviors:

- **Links:** Targeting links to specific frames is supported. When a link loads in an existing frame, the browser preserves the other frames on the page and updates the content for the specified frame.
- **Scripts:** The browser supports JavaScript interaction between frames, provided the frames have the same host, port, and protocol (HTTP or HTTPS). Interaction between frames includes the updating of multiple frames by JavaScript, the execution of JavaScript functions in other frames, and access to JavaScript variables in other frames.
- **Nested framesets:** The browser supports nested framesets and will render frames in the order in which they are encountered; however, the browser limits the depth of nested frameset documents. Frames that exceed the depth limit will be ignored.

## Creating an XHTML-MP page

The following sample creates a login page that is XHTML-MP–compliant and appears properly on the browser. This sample creates a registration page for a fictitious stock-monitoring push service. The page lets customers of an online broker service identify a stock that is of interest and set the performance conditions that must be met for the Stock Monitor service to push out a notification.

To create an XHTML-MP page, complete the procedures that follow in the order in which they appear.

The complete code sample is provided at the end of this section. See “Code sample: Creating an XHTML-MP web page” on page 49 for more information.



**Note:** In this section, each tag appears on a new line. When content exceeds the length of a line, the content continues on the next line with an irregular indentation to indicate the line break.

The complete sample should resemble the following diagram:

Completed sample, as viewed on a BlackBerry 7780 Wireless Handheld™

### Create the skeleton HTML page

1. Create a new XHTML file with the name stock-monitor.xhtml.
2. Include the doctype specification and the <html> and <head> tags for your file.
 

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Mobile 1.0//EN"
    "http://www.wapforum.org/DTD/xhtml-mobile10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <base href="http://localhost:8080/">
```
3. To help users identify their location, include the site name in the title bar.
 

```
<title>Stock Monitor -- Registration</title>
```
4. Close the <head> tag.
 

```
</head>
```
5. Add the opening and closing <body> tags.
 

```
<body>
```

```
</body>
```

6. Close the `<html>` tag.

```
</html>
```



**Note:** The browser also supports the `http-equiv="refresh"` attribute for the `<meta>` element, which you can use to redirect one page to another. For example, the following code redirects the browser to `BlackBerry/index.html` after a 2-second pause:

```
<meta http-equiv="refresh" content="2; url=http://localhost:8080/BlackBerry/index.html" />
```

## Design a navigation bar with links



**Tip:** An easily accessible navigation bar is an important part of creating an effective web page for BlackBerry devices. Users should be able to access the primary links on each page with minimal scrolling. A bar of links at the top of each page minimizes scrolling.

1. Design the page navigation by adding links at the top of the page, between the `<body>` and `</body>` elements. In this example, the navigation is nested inside a `<div>` element so that it can be formatted using a style sheet.

```
<div class="navigation">
  <a href="index.xhtml">Home</a> |
  <a href="symbols.xhtml">Query symbols</a> |
  <a href="exchanges.xhtml">Query exchanges</a> |
  <a href="monitor-help.xhtml">Help</a>
</div>
```

2. To emulate the BlackBerry interface separator, add an `<hr/>` element after the navigation bar. A properly placed `<hr/>` tag creates a non-selectable separator line on your page, which makes your web pages consistent with standard BlackBerry applications.

```
<hr/>
```



**Tip:** For a small screen, use separators to group related information and create a better visual flow for your web documents.

3. Add instructional text in a `<p>` tag.

```
<p>Welcome to Stock Monitor push application. To register, please complete
the form below: </p>
```



**Note:** To make sure that the page conforms to XHTML Basic, tags such as `<p>` must be closed.

## Create a form to collect information from the user

1. Under the instructional text, add the form. The following example invokes a `.php` script, when submitted:

```
<form name="monitor" method="post" action="stock-monitor.php">
</form>
```

2. Between the `<form>` and `</form>` elements, add a two-column table.

```
<table>
</table>
```

3. Between the `<table>` and `</table>` elements, add a row that contains an input of type “text” for the user’s email address. Because email addresses can often be long, it should span both columns of the table. The browser supports the spanning of columns and rows.

```
<tr>
```

```

        <td colspan="2">
            Type your email address:
            <input type="text" name="email" size="40" />
        </td>
    </tr>

```

4. Add a row containing the text input for the stock's trading symbol.

```

    <tr>
        <td width="45%">
            Product symbol:
        </td>
        <td width="55%">
            <input type="text" name="symbol" size="6" />
        </td>
    </tr>

```

5. Add a row containing a drop-down list that lets users select the exchange on which the stock is listed. Omit the size attribute of the <select> element to display the radio buttons as a drop-down list instead of a list box.

```

    <tr>
        <td>
            Listed on:
        </td>
        <td>
            <select name="exchange">
                <option>TSE</option>
                <option>NYSE</option>
                <option>NASDAQ</option>
                <option>Dow Jones</option>
            </select>
        </td>
    </tr>

```

6. Add a row that contains a group of radio buttons that allow users to choose whether they are notified when the stock price changes by a specified amount or when the stock price hits a high or low target price.

```

    <tr>
        <td>
            Notification trigger:
        </td>
        <td>
            <input type="radio" name="notify" value="change" checked>
                Target price change
            </input><br/>
            <input type="radio" name="notify" value="high-low">
                Target high/low values
            </input>
        </td>
    </tr>

```

7. Add a row containing a series of check boxes that lets users select items that are included in the notification, such as a chart or news items related to the selected stock.

```

    <tr>
        <td>
            Include with push:
        </td>
        <td>

```

```

        <input type="checkbox" name="chart">Performance chart</input><br/>
        <input type="checkbox" name="articles">Related articles</input>
    </td>
</tr>

```

8. Create a submit button and add a reset button to lets users reset all values to the defaults.

```

<input type="submit" name="submit" value="Register">
<input type="reset" value=" Reset " >

```

9. Add three hidden inputs for user-supplied values. These values are set through a series of JavaScript dialog boxes.

```

<input type="hidden" name="change" value="" />
<input type="hidden" name="maxtarget" value="" />
<input type="hidden" name="mintarget" value="" />

```

## Add graphics



**Tip:** When you create pages to be viewed on wireless devices, avoid using unnecessary images. A recognizable logo is one image you should include, so that visitors will know that they have arrived at the correct page.

1. Include a right-aligned image using the `<img>` tag.

```



```

2. Include an `alt` attribute to provide a short description for the image.

```



```

3. Make the image a link by nesting it in an `<a>` element.

```

<a href="www.blackberry.com">
    
</a>

```

## Create a footer with email and phone links

1. To create a footer that resembles the header, include the footer elements in a `<div>` tag, with the same navigation CLASS attribute that is used for the header navigation. The sample separates the footer with preceding and following `<hr/>` tags.

```

<hr/>
<div class="navigation">

</div>
<hr/>

```

2. Add an email link to your page.

```

<a href="mailto:info@bb-broker.com">Email</a>

```

3. Add a phone link to your page.

```

<a href="tel:416-555-0000">Telephone</a>

```

## Add styles

1. Create styles using WAP CSS syntax. These styles can be nested under a `<style>` element that is located between the `<head>` and `</head>` elements or included in a separate CSS file.

```
div.navigation {
    background-color: lightgrey;
    font-family: arial;
    font-size: 4pt;
}
```

2. If you have created an external style sheet, between the `<head>` and `</head>` elements, add a `<link>` element. This element must have attributes that define the name of the CSS file and identify the content as style properties.

```
<link rel="stylesheet" href="stock-monitor.css" type="text/css" />
```

The complete CSS file is provided at the end of this section. See "stock-monitor.css" on page 53 for more information

## Add JavaScript(s) to extend functionality



**Note:** The sample code uses an internal JavaScript to make sure that the form is not submitted with empty input fields. It also uses prompt dialog boxes to acquire additional information from users based on their choices in the form.

1. Between the `<head>` and `</head>` elements, add a `<script>` element and define the script language as JavaScript.

```
<script type = "text/javascript">
</script>
```

2. Between the `<script>` and `</script>` elements, define a function called `ValidateForm()`.

```
function ValidateForm() {
```

3. Add a routine to verify that the user has specified an email address. If no email address has been specified, display a dialog box.

```
    if ((emailID.value==null)|| (emailID.value=="")) {
        alert("Please Enter your Email Address");
        emailID.focus();
        return false;
    }
```

4. Add a routine to verify that the user has specified a stock symbol. If no stock symbol has been specified, display a dialog box.

```
    if ((symbol.value==null) || (symbol.value=="")) {
        alert("Please enter a symbol for the financial product you want
            to monitor.");
        symbol.focus();
        return false;
    }
```

5. Add a routine to determine if the user selected the Target change value notification trigger. If so, display a dialog box that lets the user specify the amount by which the stock price must change to trigger a notification message.

```
    if (document.monitor.notify[0].checked) {
```

```

        change = prompt("Specify the value change for " + symbol.value +
            " required to trigger a notification: ", change);
        if (confirm("You will be notified when " + symbol.value + " changes by
            at least $" + change + " per share, based on its value
            at the time of registration")==true) {
            document.monitor.change.value = change;
            return true;
        } else {
            return false;
        }
    }
}

```

6. Add a routine to determine if the user selected the Target high/low value notification trigger. If the user has selected the check box, display a dialog box that lets the user specify a target high followed by a dialog box that lets the user specify a target low.

```

if(document.monitor.notify[1].checked) {
    maxtarget = prompt("Please enter the target high value for " +
        symbol.value + ":", maxtarget);
    mintarget = prompt("High value set at $" + maxtarget + ". Please
        enter the target low value:", mintarget);
    if (confirm("You will be notified when the financial product " +
        symbol.value + " reaches a high of $" + maxtarget + " or
        a low of $" + mintarget + " per share.")==true) {
        document.monitor.maxtarget.value = maxtarget;
        document.monitor.mintarget.value = mintarget;
        return true;
    } else {
        return false;
    }
}

```

7. Add an onSubmit event handler to the <form> element, which calls the ValidateForm() JavaScript function created in step 1.

```

<form name="monitor" method="post" action="bb-broker.php"
    onSubmit="return ValidateForm()">

```

## Set up a queue for offline form submissions



**Note:** In the sample, offline form submission is set up by adding hidden <input> elements to the form.

- > Between the <form> and </form> elements, add a hidden <input> element for each header that you want to define. The x-rim-request-title property must be defined; other queuing parameters are optional. See "Defining queues for offline form submission" on page 33 for more information about the available queuing parameters.

For each <input> element, the name attribute corresponds to the header name (for example, x-rim-queue-id), while the value attribute defines the value for that header.

```

<input type="hidden" name="x-rim-queue-id" value="Register" />
<input type="hidden" name="x-rim-request-title" value="Stock Monitor
    Registration" />
<input type="hidden" name="x-rim-next-target" value="success.html" />

```



See “Defining queues for offline form submission” on page 33 for more information about the headers that are used to define form queues.



**Tip:** You can also set up forms by formatting your web site with the necessary headers. See “Defining queues for offline form submission” on page 33 for more information.

## Test the page

When you design content for the browser, you should test content in the browser throughout the design and creation process to verify that the page displays and functions properly on BlackBerry devices. If you wait until the page is complete before you test it, errors can be more difficult to identify and correct.

Test content using a BlackBerry device or by using the BlackBerry Device Simulator. The BlackBerry Device Simulator is included with the BlackBerry® Java™ Development Environment (BlackBerry JDE).

The simulator’s browser application includes a built-in WAP gateway, so that you can test XHTML pages by opening them in the WAP Browser configuration. You can also simulate the network gateways for the BlackBerry Browser and the Internet Browser configurations using the BlackBerry MDS Simulator. This simulator includes the content transcoding and optimization services so that your code is optimized for viewing in these browser configurations.

See “Testing web pages” on page 99 for more information.

## Code sample: Creating an XHTML-MP web page

### Example: stock-monitor.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Mobile 1.0//EN"
"http://www.wapforum.org/DTD/xhtml-mobile10.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Stock Monitor -- Registration</title>

    <!-- Adding a meta tag define how this page may be cached. -->
    <meta http-equiv="cache-control" content="public" />

    <!-- Linking to an external stylesheet. -->
    <link rel="stylesheet" href="stock-monitor.css" type="text/css" />

    <!-- Creating a javascript which checks input and selections, and requests
    further information through dialogs. -->
    <script type = "text/javascript">

      function ValidateForm() {

        <!-- set variables -->
        var emailID=document.monitor.email;
        var symbol=document.monitor.symbol;
        var change="";
        var targetmax="";
        var targetmin="";
```

```

<!-- If no email address has been entered, alert user. -->
if ((emailID.value==null)|| (emailID.value=="")) {
    alert("Please Enter your Email Address");
    emailID.focus();
    return false;
}

<!-- If no symbol has been entered, alert user. -->
if ((symbol.value==null) || (symbol.value=="")) {
    alert("Please enter a symbol for the financial product you want
        to monitor.");
    symbol.focus();
    return false;
}

<!-- If target price change used, get change amount. -->
if(document.monitor.notify[0].checked) {
    change = prompt("Specify the price change for " + symbol.value
        + " required to trigger a notification: ",
        change);
    if (confirm("You will be notified when " + symbol.value +
        " changes by at least $" + change + " per
        share, based on its value at the time of
        registration.")==true) {
        document.monitor.change.value = change;
        return true;
    }
}

<!-- If target max/min values used, get them. -->
if(document.monitor.notify[1].checked) {
    maxtarget = prompt("Please enter the target high value for " +
        symbol.value + ":", maxtarget);
    mintarget = prompt("High value set at $" + maxtarget + ". Please
        enter the target low value:", mintarget);
    if (confirm("You will be notified when the financial product "
        + Symbol.value + " reaches a high of $"
        + maxtarget + " or a low of $" + mintarget
        + " per share.")==true) {
        document.monitor.maxtarget.value = maxtarget;
        document.monitor.mintarget.value = mintarget;
        return true;
    } else {
    }
}
}
</script>

</head>

<body>

    <!-- Creating a navigation bar. -->
    <div class="navigation">
        <a href="index.xhtml">Home</a> |
        <a href="symbols.xhtml">Query symbols</a> |

```

```

    <a href="exchanges.xhtml">Query exchanges</a> |
    <a href="monitor-help.xhtml">Help</a>
</div>
<hr/>

<p>Welcome to Stock Monitor push application. To register, please complete
    the form below: </p>

<!-- Creating a form to collect information. -->
<form name="monitor" method="post" action="stock-monitor.php"
    onSubmit="return ValidateForm()">
    <hr/>
    <!-- Adding a table to organize form elements. -->
    <table>
        <tr>
            <td colspan="2">
                Enter your email address:
                <input type="text" name="email" size="40" />
            </td>
        </tr>
        <tr>
            <td width="45%">
                Product symbol:
            </td>
            <td width="55%">
                <input type="text" name="symbol" size="6"/>
            </td>
        </tr>
        <tr>
            <td>
                Listed on:
            </td>
            <td>
                <!-- Creating a drop-down list. -->
                <select name="exchange">
                    <option>TSE</option>
                    <option>NYSE</option>
                    <option>NASDAQ</option>
                    <option>Dow Jones</option>
                </select>
            </td>
        </tr>
        <tr>
            <td>
                Notification trigger:
            </td>
            <td>
                <!-- Creating a group of radio buttons. -->
                <input type="radio" name="notify" value="change" checked>
                    Target price change
                </input><br/>
                <input type="radio" name="notify" value="high-low">
                    Target high/low value
                </input>
            </td>
        </tr>
    </table>

```

```

        <tr>
            <td>
                Include with push:
            </td>
            <td>
                <!-- Creating a series of checkboxes -->
                <input type="checkbox" name="chart">
                    Performance chart
                </input><br/>
                <input type="checkbox" name="articles">
                    Related articles
                </input>
            </td>
        </tr>
    </table>
    <hr/>
    <!-- Adding submit and reset buttons. -->
    <center>
        <input type="submit" name="submit" value="Register" />
        <input type="reset" value="Reset" />
    </center>

    <!-- Adding some hidden inputs that will be set via the ValidateForm()
         javascript function. -->
    <input type="hidden" name="change" value="" />
    <input type="hidden" name="maxtarget" value="" />
    <input type="hidden" name="mintarget" value="" />

    <!-- Adding some hidden inputs that define the queue for the offline
         submission of the form. -->
    <input type="hidden" name="x-rim-queue-id" value="Register" />
    <input type="hidden" name="x-rim-request-title" value="Stock Monitor
        Registration" />
    <input type="hidden" name="x-rim-next-target" value="success.html" />

</form>

<!-- Adding an image with a link. -->
<a href="www.blackberry.com">
    
</a>

<!-- Adding footer with non-browser links. -->
<hr/>
<div class="navigation">
    Contact Stock Monitor:
    <a href="mailto:info@bstock-monitor.com">Email</a> |
    <a href="tel:416-555-0000">Telephone</a>
</div>
<hr/>

</body>
</html>

```

---

**Example: stock-monitor.css**

```
p {
  font-family: arial;
  font-size: 6pt;
  colour: black
}
table {
  background-color: lightgrey;
  font-family: arial;
  font-size: 6pt;
}
input {
  -wap-input-required: true;
  font-family: arial;
  font-size: 6pt;
}
select {
  font-family: arial;
  font-size: 6pt;
  border-width: 0pt;
}
div.navigation {
  background-color: lightgrey;
  font-family: arial;
  font-size: 4pt;
}
a {
  color: darkblue
}
```

---



# Creating WML pages

Using WML

Creating a WML page

Code sample: Creating a WML web page

## Using WML

WML is an XML language that conforms to XML 1.0 rules; for example, tags are case-sensitive, and every opening tag must either have a corresponding closing tag or be self-closing. The browser supports WML 1.3 and WML Script 1.2.1. See “WML language reference” on page 123 for more information about supported WML tags.



**Note:** The browser supports WML Script except for the `Float` standard library and all functions that use floating point values. Floating point refers to numbers with a varying number of digits before or after the decimal point; that is, the decimal point can float.

WML is primarily a text-based language that facilitates chunking information into small sections for display on mobile devices. Because of the small screen size, typical desktop-oriented HTML pages do not typically display effectively on a mobile browser. HTML content is generally designed for a much larger space, and it often includes elements such as complex tables or framesets that render poorly or not at all on the browser. Also, the amount of text on most HTML sites results in a browsing experience that requires a great deal of scrolling.

WML breaks down information into chunks, called “cards.” Each card contains a single topic, or about a screen’s worth of content. These related WML cards are then connected using navigational links. A set of related WML cards are included in a single .wml file, which is referred to as a “deck.” From an HTML perspective, a WML deck is similar to creating a single HTML file that includes multiple `<body>` sections.

WML offers several advantages for mobile browsers:

- WML decks reduce the number of client-server transactions; decks are transmitted as individual pages.
- WML cards provide an optimal device browsing experience because chunks of content display as a single screen and they require minimal scrolling.
- WML has a small tagset compared to HTML.
- WML supports forms, option lists, and text entry, which is ideal for users who want to retrieve specific information.

## WML design tips

- **Structure:** Limit one element for each line of code.
- **Syntax:** Use proper syntax. WML is an XML language; therefore, content must be well-formed.
  - Every opening tag—including standalone tags such as `<p>`, `<br>`, and `<img>`—must either have a corresponding closing tag or be self-closing.

- Elements and attributes must be lowercase.
- Attributes must be in quotations.
- Elements must be properly nested such that no element overlaps another element.
- **File types:** Verify that your application server accepts .wml files, plus any other file types (for example, .gif, .pl) that might be included in your content. Application servers include a list of supported file and application types, known as MIME types.
- **WML-specific elements:** While the WML tagset is essentially a subset of HTML, several elements are unique to WML.

- WML `<do>` elements are triggered by user-initiated events. Each `<do>` element has a corresponding action that occurs when the user performs a `<do>` event.

In the browser, the first two `<do>` elements that are defined on a card appear both as menu items on the browser menu and as "soft keys" in a non-scrolling area at the bottom of the screen. Any additional `<do>` elements are added to the browser menu.

If a `label` attribute is included in the `<do>` element, the label text is displayed as the menu item and the soft key. If no label is included, a default label is displayed based on the type of `<do>` element, such as **Accept, Prev, Help, Reset, Options, or Delete**.

- WML `<select>` items are displayed either as a list of radio buttons (for single selection) or as a list of check boxes (for multiple selection). Users can use the trackwheel or keyboard to scroll through the options, and then click the **Select Option** menu item or press the Space or Enter key to select an option.

You can use the `onpick` attribute to perform an action when the user selects an option. See "Use WML events" on page 60 for more information.

- WML `<access>` items limit which other web pages can reference your pages. To grant an authorized domain access to your page (for example, `www.blackberry.com`), provide the `domain` attribute.

```
<head>
  <access domain="255.255.255.255"/>
</head>
```

When a user attempts to access the page from another domain, the "Access denied" message appears.



**Note:** Specifying deck access limits users' ability to open your web site in a browser. Unless your browser application is designed for a private audience (such as an intranet application), do not use the `<access>` element.

## Creating a WML page

The following example demonstrates how to create a simple login page in WML. It also provides some tips about how to create a page that displays successfully in the browser.

The following sections walk through the steps required to create a sample WML deck. In this example, a deck is created that contains three cards: a login screen, a links screen, and a help screen to assist users who are having trouble logging in. The complete WML sample is provided at the end of this section.

### Define the WML deck and cards

1. Create a file with the name `stock-monitor.wml`.



2. Define your document.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
```

3. Insert the opening WML element tag. All templates and cards are added between this tag and its closing tag.

```
<wml>
```

4. Create the main card that users will use to log in. This card also functions as the Home page of the site.

```
<card id="Page1" title="Stock Monitor Registration">
```

5. Specify a default <do> action for the card. <do> elements with the type="accept" attribute are the default card action and they appear first on the browser menu. The browser automatically performs the first <do> element with the type accept when the user has provided input for all input elements on a card and presses the ENTER key.

```
<do type="accept" label="Next">
```

Provide a descriptive label for each <do> action. Limit the length of <do> labels to 12 characters so that the label does not exceed the width of the browser menu. Capitalize the first letter of each word in a label to maintain consistency with BlackBerry menu items. This page lets users continue to the next card in the deck, so it is labelled "Next."

All <do> elements require a corresponding action, for example, <go> or <prev>. A subsequent procedure adds an action to this element.

6. Add the closing tag for this card.

```
</card>
```

7. Create additional cards with Next and Back soft keys. This card contains an option list of links.

```
<card id="Page2" title="Stock Monitor -- Choose Stock">
  <do type="accept" label="Next">
    <go href="#Page3" />
  </do>
  <do type="prev" label="Back">
    <go href="#Page1" />
  </do>
</card>
```

8. Create a new card for the help page.

```
<card id="Help" title="Help">
  <p>
    This page contains helpful hints.
  </p>
</card>
```

9. Add the closing tag for this deck.

```
</wml>
```

## Add a template

WML templates let you include content in several cards at the same time. Using templates saves time and minimizes the content that is sent over the wireless network; content that is duplicated across multiple cards only needs to be included once in the deck. Templates can contain only <do> and <onevent> tags.

In this example, we want to add a help menu item that is available for each card.

1. Create the template by inserting the following tag immediately after the `<wml>` tag:

```
<template>
```

2. Add the repeated content or functionality. To add a Help menu item, add a `<do>` element that contains a `<go>` action that identifies the help page to be displayed:

```
<do type="help" label="Help" >
  <go href="#help"/>
</do>
```



**Notes:**

- To create a link to a subsidiary card, prefix the card id with a number sign (#) character.
- WML `<do>` elements appear at the bottom of the screen, from left to right. The order in which the items appear depends on the sequence of `<do>` elements in the file; the first `<do>` element appears at the left.

3. Close the `<template>` element:

```
</template>
```

The Help soft key is added by default to every card in the deck. Because most cards in the deck contain Next and Back soft keys, however, the Help option will appear only as menu item.

See “Use WML events” on page 60 for more information about overriding template content on specific cards.

## Add input fields

1. Add an email input field to the first card. WML does not require that you enclose `<input>` elements in a `<form>`.

```
<p>
  Please enter your email address:
  <input type="x-rim-email-input" name="email" />
</p>
```



**Note:** The browser hides user input for fields in which you specify `type="password"`.

By default, input fields begin on a new line. Text fields span the width of the browser window. To limit the number of characters that users can type, use the `maxlength` attribute.

2. Add a stock symbol input field to the second card.

```
<p>
  Product trading symbol: <input type="text" name="symbol"
                        format="*AAA" />
</p>
```

3. To specify the action that the card performs when the user submits the content, add a `<go>` tag inside the primary `<do>` element that you created in the “Define the WML deck and cards” procedure. In this example, a Perl script runs. In the following code, the `<do>` element already exists.

```
<do type="accept" label="Submit">
  <go href="/cgi-bin/loginScript.pl" method="post">
    <postfield name="name" value="$(loginID)" />
    <postfield name="password" value="$(password)" />
  </go>
```

```
</do>
```

Using the `<do>` element, instead of `<a>` and `<anchor>` tags lets you bind an action to the default option for the page.

## Create an option list

1. Add introductory text to the Page2 card.

```
<p>
  Listed on:
</p>
```

2. Add an option list using the `<select>` element.

```
<select name="name" title="Listed on" value="TSE">
  <option value="TSE">TSE</option>
  <option value="NYSE">NYSE</option>
  <option value="NASDAQ">NASDAQ</option>
  <option value="Dow Jones">Dow Jones</option>
</select>
```



**Note:** This section adds a single-selection option list (a set of radio buttons). To render a list as a series of check boxes, add the attribute `multiple="true"` to the `<select>` element.

3. On the Page3 card, add two more option lists. One will let users specify the notification trigger method; the other will let them select what additional information will be pushed with the notification. The second list is a multiple-selection list.

```
<p>
  Notification trigger:
  <select name="trigger" value="c">
    <option value="c">Target price change</option>
    <option value="r">Target high and low values</option>
  </select>
</p>
<br/>
<p>
  Include with push:
  <select name="includes" multiple="true">
    <option value="chart">Performance chart</option>
    <option value="articles">Related articles</option>
  </select>
</p>
```

## Add an image

Use the `<img>` tag to include an image on the Help page. Place the image right-aligned in a new paragraph after the Help page contents.

```
<p align="right" >
  <!-- Adding an image with a link. -->
  <a href="www.blackberry.com">
    
  </a>
</p>
```

## Use WML events

This section describes how to use WML events. You can use the following WML events:

- *User-initiated events* trigger actions when the user interacts with a screen element. You can use the following events:
    - `onenterbackward`: Triggers an action to occur when the user enters the card by clicking `<prev>`
    - `onenterforward`: Triggers an action to occur when the user enters the card by clicking `<go>`
    - `onpick`: Triggers an action to occur when the user selects an option
  - *Timeline events* trigger actions that occur at a certain point in time. A single event is available:
    - `ontimer`: Triggers an action to occur when the timer on a card expires
1. On the `Help` card, add an `onpick` event attribute to each option in the option list, so that the browser opens the specified URL when the user selects the option.

```
<option value="rim" onpick="http://www.rim.com">
    Research In Motion
</option>
<option value="blackberry" onpick="http://www.blackberry.com">
    BlackBerry products
</option>
<option value="developers" onpick="http://www.blackberry.com">
    BlackBerry Developer Zone
</option>
```

2. Modify the `<card>` element for the first card (`Page1`) to add a timer event, so that the `Help` card opens if the user takes longer than 2 minutes to log in.

```
<card id="main" title="BlackBerry WML sample" ontimer="#help">
    <timer value="2400"/>
```

Use the WML `ontimer` attribute to force new content to the browser after a specified amount of time has passed.

The `timer` value is measured in tenths of a second, and the counter is reset when the page is refreshed. The `<timer>` element is commonly used to redirect a WML page to another WML page; setting the `value` attribute to 1 redirects the page automatically.

## Add WMLScript(s) to extend functionality

1. Create a file with the name `stock-monitor.wmls`.
2. Define a function called `setTriggers()`.
 

```
function setTriggers() {
```
3. Declare variables for each trigger. A variable must be declared for both the target high and low options. Set each variable to an initial value of null.
 

```
    var valChange = "";
    var valHigh = "";
    var valLow = "";
```
4. Retrieve the trigger variable from the `Page3` card.
 

```
    var trigger = WMLBrowser.getVar("trigger");
```

- Determine which trigger option was selected.

```
var change = String.find(trigger,"c");
var range = String.find(trigger,"r");
```

- If the Target price change option was selected, display a dialog box that lets the user specify the amount by which the stock price must change to trigger a notification message.

```
if(change) {
    var valChange = Dialogs.prompt("Change value: ","");
    var message = "You will be notified when the value changes by $"
        + valChange;
}
```

- If the Target high/low value option was selected, display a dialog box that lets the user specify a target high, followed by a dialog box that lets the user specify a target low.

```
if(range) {
    var valHigh = Dialogs.prompt("High value: ","");
    var valLow = Dialogs.prompt("Low value: ","");
    var message = "You will be notified when the value reaches a high of $"
        + valHigh + " or a low of $" + valLow;
}
```

- Add a go event to the Page3 card that calls the setTriggers WMLScript function created in step 1.

```
<do type="accept" label="Next">
    <go href="stock-monitor.wmls#setTriggers()" />
</do>
```

## Set up a queue for offline submissions

- Between the <go> and </go> elements, add a <postfield> element for each header that you want to define. The x-rim-request-title property must be defined; other queuing parameters are optional.

For each <input> element, the name attribute corresponds to the header name (for example, x-rim-queue-id), while the value attribute defines the value for that header.

```
<postfield name="x-rim-queue-id" value="Login"/>
<postfield name="x-rim-request-title" value="Login"/>
<postfield name="x-rim-next-target" value="success.wml"/>
```

See “Defining queues for offline form submission” on page 33 for more information about the headers used to define form queues.



**Note:** You can also set up forms by formatting your web site with the necessary headers. See “Defining queues for offline form submission” on page 33 for more information.

## Override a template

You might not want template elements to appear on every card in the deck. Use a technique called shadowing to hide certain template elements on a card by adding the <noop> element. For example, you can hide the link to the Help card on the Help card itself. To use the <noop> element to hide a soft key, first add the <do> element that you want to hide, and then add the <noop> element.

- On the Help card, after the <card> element, add the <do> element with the label="Help" attribute from the template that you want to shadow.

```
<do type="help" label="Help" >
```

2. Instead of including the `<go>` element within the `<do>` element, use the `<noop/>` element. `<noop/>` tells the browser to hide this option for this card.

```
<noop/>
</do>
```

## Test the page

When you design content for the browser, you should test content in the browser throughout the design and creation process to verify that the page displays and functions properly on BlackBerry devices. If you wait until the page is complete before you test it, errors can be more difficult to identify and correct.

Test content using a BlackBerry device or by using the BlackBerry device simulator. The BlackBerry device simulator is included with the BlackBerry JDE.

The simulator's browser application includes a built-in WAP gateway, so that you can test WML pages by opening them in the WAP Browser configuration. You can also simulate the network gateways for the BlackBerry Browser and the Internet Browser configurations using the BlackBerry MDS Simulator. This simulator includes the content transcoding and optimization services so that your code is optimized for viewing in these browser configurations.

See "Testing web pages" on page 99 for more information.

## Code sample: Creating a WML web page

### Example: login.wml

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/
wml_1.1.xml">

<!--Copyright (C) 2004 Research In Motion Limited. All Rights Reserved -->
<!--loginScript.pl script, referenced in this example, is not shown here -->

<wml>

    <template>
        <do type="help" label="Help">
            <go href="#Help"/>
        </do>
    </template>

    <card id="Page1" title="Stock Monitor Registration" ontimer="#Help">
        <timer value="2400"/>
        <do type="accept" label="Next">
            <go href="#Page2" />
        </do>

        <p>
            Please enter your email address:
            <input type="text" name="email" />
        </p>
        <p align="right" >
            <!-- Adding an image with a link. -->
            <a href="www.blackberry.com">
```

```

        
    </a>
</p>
</card>

<card id="Page2" title="Stock Monitor -- Choose Stock">
    <do type="accept" label="Next">
        <go href="#Page3" />
    </do>
    <do type="accept" label="Back">
        <go href="#Page1" />
    </do>
<p>
    Product trading symbol: <input type="text" name="symbol"
        format = "*AAA" />
</p>
<p>
    <!-- Creating a single selection list. -->
    Listed on:
    <select name="name" title="Listed on" value="TSE">
        <option value="TSE">TSE</option>
        <option value="NYSE">NYSE</option>
        <option value="NASDAQ">NASDAQ</option>
        <option value="Dow Jones">Dow Jones</option>
    </select>
</p>
<p align="right" >
    <!-- Adding an image with a link. -->
    <a href="www.blackberry.com">
        
    </a>
</p>
</card>

<card id="Page3" title="Stock Monitor -- Push Options">
    <do type="accept" label="Next">
        <go href="stock-monitor.wmls#setTriggers()" />
    </do>
    <do type="accept" label="Back">
        <go href="#Page2" />
    </do>
<p>
    <!-- Creating a single selection list. -->
    Notification trigger:
    <select name="trigger" value="c">
        <option value="c">Target price change</option>
        <option value="r">Target high and low values</option>
    </select>
</p>
<br/>
<p>
    <!-- Creating a multiple selection list. -->
    Include with push:
    <select name="includes" multiple="true">
        <option value="chart">Performance chart</option>
        <option value="articles">Related articles</option>
    </select>
</p>

```

```

    <p align="right" >
      <!-- Adding an image with a link. -->
      <a href="www.blackberry.com">
        
      </a>
    </p>
  </card>

  <card id="Page4" title="Confirm Registration Data">
    <p>
      You chose to monitor $(symbol), listed on $(exchange)<br/>
      $(message)
    </p>
    <br/>
    <p align="right" >
      <!-- Adding an image with a link. -->
      <a href="www.blackberry.com">
        
      </a>
    </p>
    <do type="accept" label="Submit">
      <go href="stock-monitor.php" method="post">

        <!-- Adding some postfields for each variable that must be
          submitted. -->
        <postfield name="symbol" value="$(symbol)"/>
        <postfield name="exchange" value="$(exchange)"/>
        <postfield name="valChange" value="$(valChange)"/>
        <postfield name="valHigh" value="$(valHigh)"/>
        <postfield name="valLow" value="$(valLow)"/>
        <postfield name="includes" value="$(includes)"/>

        <!-- Adding some postfields that define the queue for
          the offline submission of the form. -->
        <postfield name="x-rim-queue-id" value="Register"/>
        <postfield name="x-rim-request-title" value="Stock Monitor
          Registration"/>
        <postfield name="x-rim-next-target" value="success.wml"/>
      </go>
    </do>
    <do type="accept" label="Back">
      <go href="#Page3"/>
    </do>
  </card>

  <card id="Help" title="Help">

    <!-- Disabling the Help soft key. -->
    <do type="help" label="Help">
      <noop/>
    </do>
    <p>
      This page contains helpful hints.
    </p>
    <p>
      For more information, click an option:
      <select name="URLSelection">

```



```

        <option value="rim" onpick="http://www.rim.com">
            Research In Motion
        </option>
        <option value="blackberry" onpick="http://www.blackberry.com">
            BlackBerry products
        </option>
        <option value="developers" onpick="http://www.blackberry.com">
            BlackBerry Developer Zone
        </option>
    </select>
</p>
</card>

</wml>

```

---

#### Example: stock-monitor.wmls

```

function setTriggers() {

    var trigger = WMLBrowser.getVar("trigger");
    var change = String.find(trigger,"c");
    var range = String.find(trigger,"r");
    var valChange = "";
    var valHigh = "";
    var valLow = "";

    if(change) {
        var valChange = Dialogs.prompt("Change value: ","");
        var message = "You will be notified when the value changes by " + valChange;
    }
    if(range) {
        var valHigh = Dialogs.prompt("High value: ". "");
        var valLow = Dialogs.prompt("Low value: ", "");
        var message = "You will be notified when the value reaches a high of $"
            + valHigh + " or a low of $" + valLow;
    }

    var ret = Browser.go("bb.wml#Page4");
}

```

---




# Creating browser push applications

Push applications  
The BlackBerry push process  
Defining push attributes  
RIM push service implementation  
PAP push service implementation

## Push applications

With push applications, you can send new web content and alerts to specific users. Users do not have to request or download the data because the push application delivers the information as it becomes available.

Two types of push applications exist:

- **Browser push applications:** Web content is sent to the browser on the BlackBerry device. The BlackBerry Browser configuration supports RIM and PAP push applications. WAP Browser configuration supports WAP push applications. The Internet Browser configuration does not support push applications.  
 **Note:** RIM push applications require BlackBerry Enterprise Server for Microsoft Exchange Version 3.5 or later, or BlackBerry Enterprise Server for IBM Lotus Domino Version 2.2 or later, with the BlackBerry MDS Connection Service turned on.
- **Client/server push applications:** Data is sent to a custom Java application on the BlackBerry device. Client/server push applications consist of a custom client application for the BlackBerry device and a server-side application that pushes content to the client application. This approach provides more control over the type of content that you can send out and how this data is processed and displayed on the device compared to browser push applications. See the *BlackBerry Java Development Environment Development Guide* for information about writing custom Java applications.

## BlackBerry Browser configuration push support

The BlackBerry Enterprise Server, with the BlackBerry MDS Connection Service running, manages the flow of data from the push application to the device using the same encrypted channel that is used for data communication between the device and the BlackBerry Enterprise Server. The push application sends data to users based on their email addresses.

With a central database of the BlackBerry device users in the organization, the BlackBerry Enterprise Server directs content to the appropriate devices. The network gateway manages the connection to the wireless network and verifies that content is delivered to users as soon as they are in a sufficient wireless coverage area.

Server applications push content to devices by sending HTTP POST requests to the BlackBerry Enterprise Server. The BlackBerry MDS Connection Service of the BlackBerry Enterprise Server forwards the content to the appropriate devices, based on user email addresses. On the device, a separate browser push listener thread listens on port 7874 for incoming messages and processes those messages.

The BlackBerry MDS Connection Service supports two push service implementations:

- **RIM Push:** Sends the push content as a byte stream to the destination user and port that is specified in the URL of the push message. Push data can be stored on the in stored in RAM of the BlackBerry MDS Connection Service server or in the BlackBerry Enterprise Server database.
- **Push Application Protocol (PAP):** Sends an HTTP POST request containing a PAP message. This message is a MIME multipart message that includes an XML document specifying the control entity and the push content. The control entity contains information about the destination device address, message ID, delivery time stamps, and so on.

## Supported push methods

You can push web content to the browser in one of the following ways:

- **Browser channel push:** Applications can push data to the device that creates or updates channels. These pushes appear on the device Home screen with a custom icon. Channels act as browser-based applications on the devices for regular updates of certain types of data.

A channel push request can include URLs for two icons:

- one to indicate that content is new or updated
- one to indicate that content has been read

When the device receives a channel push message, it creates a new channel, or updates the channel if the channel already exists. When a channel has been added or updated, an icon appears on the Home screen to alert users that content is available. For example, an order-tracking channel could change the icon as product orders are entered. Users click the icon to open an instance of the browser that displays the pushed content.

Channel push requests include a channel identifier that uniquely describes the channel. Push applications can delete a channel and remove the icon from the device Home screen by sending another push request with the ID of the channel to delete.



**Note:** Deleting a channel deletes only the instance of the channel that is currently stored on the BlackBerry device. If the server pushes the channel again, it reappears on the device Home screen.

- **Message push:** Applications can push content pages to the message list on the device. The message push request might include a descriptive title, which appears in the message list. Otherwise, the browser message displays the URL. Users open the message in the message list to open the browser and view the specified page.

Push applications can include the content in the browser message, so that the browser renders the pushed content immediately. Alternatively, the application can include only the URL, so that the browser retrieves the content only when the user requests it, either from the local cache or from the network.

- **Browser cache push:** Applications can push content directly to the browser cache. Pushing content to the cache lets users access the content quickly, even when they are outside a wireless coverage area. Cache push content can be associated with a channel.

If a channel identifier is specified, the content URL is added to the appropriate channel (if the channel is already active on the device) and to the persistent cache. Otherwise, the content is added to the persistent cache only. The user receives no indication that the content has been updated. The next time that the user visits the specified URL, the browser retrieves the content from the cache.

When an application pushes data to the browser cache, the application can include an expiry time that defines how long the data remains in the cache before it is cleared.

When performing a Browser Channel or Browser Content push, the default amount of time that push content will be stored in cache memory varies depending on which version of the BlackBerry Device Software is running on the device. In BlackBerry Device Software Version 3.8 and later, pushed content is cleared from the cache after 12 hours. On devices running an earlier version of the BlackBerry Device Software, pushed content expires and is cleared from the cache after 29 days.

To increase or decrease the time content stored in cache memory, specify a date and time in the HTTP header of the push request with the Expires header. The following example will store content in cache until September 7th, 2005 at 8 AM.

```
Expires: Wed, 07 Sept 2005 08:00:00 GMT
```



**Note:** If the BlackBerry device is running low on flash memory, the browser cache may be cleared to free up needed space. This can result in content being cleared from the cache prior to the value specified in the Expires header.

## Reliable pushes

Push application developers can configure their push applications to provide acknowledgement upon the successful delivery of a push submission. This acknowledgement lets them verify that content has reached the device.

The following two levels of acknowledgement are available:

- **Transport level:** The destination device sends a message to the push initiator acknowledging that the content has arrived at the destination device. Transport-level acknowledgment provides confirmation that the data has been delivered to the device; there is, however, no guarantee that the data has been delivered to the client application.

Transport-level acknowledgement is supported for all destination devices. If no acknowledgement level is specified by the push application, transport level is assumed.

- **Application level:** The destination device sends a message to the push initiator only after the content has reached the intended client application.

Application-level acknowledgement is supported only by those devices running BlackBerry Device Software Version 4.0 or later.

To maximize the reliability of push messages, push application developers can specify an **Application Level Acknowledgement Preferred** setting, which uses application-level acknowledgements for those devices running BlackBerry Device Software Version 4.0 or later, and transport-level acknowledgements for those devices running earlier versions of the BlackBerry Device Software.

## Additional push application features

Both RIM Push and PAP Push service implementations support the following tasks:

- **Push submission result notification:** Lets push application developers notify users that a push has arrived on their device.
- **Deliver-Before time stamp:** Lets push application developers provide a time stamp before which the push is required to be delivered. If a push is not successfully delivered by the specified time, the push is considered to have failed.

Only the PAP push service implementation supports the following additional tasks:

- **Deliver-After time stamp:** Lets push application developers provide a time stamp before which the push must not be delivered. If a push is not successfully delivered after the specified time, the push is considered to have failed.
- **Push cancellation:** Lets push applications cancel a push submission that has already been sent.
- **Push status query:** Lets push applications check the status of a push submission.

## WAP Browser configuration push support

For the WAP Browser, a WAP Push service record must be provisioned on the BlackBerry device to push data to the device. WAP Push service records are usually sent to the device during registration.

### Supported push methods

Server applications push content to the BlackBerry device using one of the following three methods:

- **Existing WAP connections:** This option is available only when a WAP connection is open between the device and the WAP gateway.
- **SMS messages:** If an existing WAP connection is not available, the service record provisioned for the GPRS and CDMA networks typically uses Short Message Service (SMS).

In addition, wireless service providers can restrict incoming SMS messages to specific sources. The source address restrictions are specified as parameters in the WAP Push service record that is sent during registration.

- **UDP messages:** If an existing WAP connection is not available, the service record provisioned for the iDEN network typically uses UDP.

On the BlackBerry device, the WAP Push service record contains information about how the device can receive WAP pushes. The service record also specifies on which ports the WAP Push Processor listens for incoming WAP Push messages and how those incoming messages are managed.

### Push message types

WAP Push accepts the following two message types:

- **Service indications:** These are self-contained messages with some text to inform the user about some event or notification. The entire text of the message is included in the service indication that is pushed to the BlackBerry device.

- **Service loadings:** These messages contain a URL that points to the real content. The service loading message is pushed to the BlackBerry device first, and then the browser automatically downloads content from the URL location.

When a push message is successfully or unsuccessfully processed by the content-specific push handler, a push completion notification is sent back to the push gateway.

By default, both service indication and service loading messages are managed by the browser automatically. Users can change how incoming push messages are managed, or turn off the browser push feature in the browser configuration properties.

## The BlackBerry push process

1. The push application sends an HTTP POST request to the BlackBerry MDS Connection Service on the web server listen port. The default port number is 8080 for the BlackBerry MDS Simulator and IBM Lotus Domino, and 8300 for Microsoft Exchange.
2. The BlackBerry MDS Connection Service sends an acknowledgement to the push application.
3. The BlackBerry MDS Connection Service closes the connection.
4. The BlackBerry MDS Data Optimization Service converts the content of the request, if necessary, and sends it to the BlackBerry Enterprise Server, which compresses and encrypts the content before sending it to the specified BlackBerry devices.
5. The specified BlackBerry devices receive the pushed content. The browser listens on port 7874 to receive the data.
6. The specified BlackBerry devices return an acknowledgement to the BlackBerry MDS Connection Service.



**Note:** System administrators set a **Flow control timeout** parameter for BlackBerry MDS Connection Service that defines the length of time that the service waits for the BlackBerry device to return an acknowledgement before it deletes pending data.

The push application does not receive an error message if the pushed data does not reach the BlackBerry device.

## Defining push attributes

You must define two sets of attributes for a browser push:

- attributes that tell the BlackBerry MDS Connection Service what to push and to whom, the reliability level, priority, delivery time stamps, and so on
- attributes that tell the browser what to do with the content once it arrives

### BlackBerry MDS Connection Service push attributes

RIM and PAP push service implementations define push attributes (for example, reliability, priority, delivery time stamps) in different ways:

- RIM push applications define push attributes using HTTP headers.

- PAP push applications include an XML document that specifies the control entity and the push content. The control entity contains information about the destination device address, message ID, delivery time stamps, and so on.

## BlackBerry MDS Connection Service push attributes: RIM push application HTTP headers

Each pushed message can include the content (or a pointer to the content) and several attributes. Attributes are encoded as HTTP headers.

The following headers can be included with a pushed message:

- X-Rim-Push-ID: *unique\_id*  
Specifies a string that uniquely identifies the message.  
Typically, you should specify a URL with a value, such as 123@foo.rim.com.
- X-Rim-Push-Description: *text\_description*  
Specifies a string that provides a brief description of the push application.
- X-Rim-Push-Priority: high | medium | low | none  
Specifies what kind of notification users receive when content is pushed to the device.
  - High: The user is notified using the user-specified Browser Message Profile notification (for example, a ring tone or vibration). In addition, an unread icon is added to the Home Screen, and a dialog box is displayed, which states that a new push has been received.
  - Medium: The user is notified using the user-specified Browser Message Profile notification (for example, a ring tone or vibration) and an unread icon is added to the Home screen.
  - Low: The user is notified using the user-specified Browser Message Profile notification (for example, a ring tone or vibration) and an unread icon is added to the Home Screen.
  - None: An unread icon is added to the Home Screen.
- X-Rim-Push-Reliability: transport | application | application-preferred  
Specifies the delivery reliability level of the content.
  - Transport: The default reliability level. Sends a message from the device acknowledging that the content has arrived.  
Transport-level acknowledgement is supported for all destination devices.
  - Application: Sends an acknowledgement message from the device only when the content has reached the intended client application.  
Application-level acknowledgement is supported only by devices that are running BlackBerry Device Software Version 4.0 or later.
  - Application-Preferred: Sends application-level acknowledgement from devices that are running at least BlackBerry Device Software Version 4.0 or later, and transport-level acknowledgement from devices running earlier versions of the BlackBerry Device Software.
- X-Rim-Push-NotifyURL: *valid\_url*  
Specifies a URL to which a result notification is sent.



- X-Rim-Push-Delete-URL: *valid\_url*

Specifies a URL the the BlackBerry Browser retrieves when the user deletes the channel on his or her BlackBerry device. The URL is not opened for the user to view; instead, this URL is retrieved in the background. By retrieving the URL, the BlackBerry device notifies the push originator that the channel no longer exists on the device.

When the browser retrieves the URL, it does not supply any parameters describing the the channel being deleted. You must therefor add parameters to the URL so that you can uniquely identify the push when the URL is requested.

X-Rim-Push-Delete-Url: `http://myserver/deleteNotify.php?pushID=45&user=54`

- X-Rim-Push-Deliver-Before: *delivery\_date*

Specifies the date and time, in HTTP format, by which the content must be delivered to the device. Content that has not been delivered before this date is not delivered.

- X-Rim-Push-Use-Coverage: *true | false*

Specifies a prompt to the BlackBerry MDS Connection Service to determine whether a target device is in network coverage. If the target device is not in a wireless coverage area, the BlackBerry MDS Connection Service will return the X-Rim-Device-State header with a value of *false* and close the push connection. When the device returns to a wireless coverage area, the BlackBerry MDS Connection Service will notify the push application, which can then resume the push process.

- *true*: The push application should attempt to push the content to the target device only if the target device is in a wireless coverage area.
- *false*: The default value. The BlackBerry Connection Service should not query the connection status of the target device.

- X-Rim-Transcode-Content: *\*/ \* | none | list\_of\_mime\_types*

Specifies which types of pushed content the server should transcode.

- *\*/ \**: The server transcodes all content.
- *none*: The server does not transcode any content.
- *list-of-mime-types*: The server transcodes content types in the given comma-separated list of MIME types. For example:

X-Rim-Transcode-Content: `image/png, image/jpeg, image/vnd.wap.wbmp`

- Content-Location: *content\_url*

Specifies a string that identifies the URL from which the content is downloaded, if the content is not included in the push.

- Content-Type: *list\_of\_mime\_types*

Specifies a comma-separated list of MIME types that can be included in the pushed content. For example:

Content-Type: `image/png, image/jpeg, image/vnd.wap.wbmp`

- Cache-Control: *no-cache | max-age | must-revalidate*

Specifies the caching parameters for the pushed content.

- **no-cache:** The content is not cached.
- **max-age:** The time, in seconds, before cached content expires.
- **must-revalidate:** The page is always retrieved again, even when going back in the History list.

## BlackBerry MDS Connection Service push attributes: PAP push application control elements and attributes

The Push Access Protocol Specification identifies several messages and responses sent between the push initiator (that is, the server-side push application) and the Push Proxy Gateway (that is, the BlackBerry MDS Connection Service).

Your PAP push application can send one of the following messages to the BlackBerry MDS Connection Service:

- push message
- cancel message
- statusquery-message

See the *Push Access Protocol (WAP-247-PAP-20010429-a)* specification for more information about writing server-side push applications using PAP. See the PAP 2.0 DTD for information about the WAP Push document type definitions (DTDs).

### PAP push message request

A push message request is sent from the push initiator to the BlackBerry MDS Connection Service, specifying the destination devices to which the specified content is sent, as well as a unique message ID and the delivery parameters.

A PAP push message request is a MIME multipart message, which consists of the following items:

- an XML document specifying the control entity
- the push content

The following is a sample push message XML control entity:

```
<pap>
  <push-message push-id="123@foo.rim.com"
    deliver-before-timestamp="2006-08-08T17:00:00Z"
    ppg-notify-requested-to="http://foo.rim.net/ReceiveNotify">
    <address address-value="WAPPUSH=john.doe%40rim.com%3A7874/TYPE=USER@rim.net" />
    <address address-value="WAPPUSH=jane.doe%40rim.com%3A7874/TYPE=USER@rim.net" />
    <address address-value="WAPPUSH=2100001A%40rim.com%3A7874/TYPE=USER@rim.net" />
    <quality-of-service delivery-method="unconfirmed"/>
  </push-message>
</pap>
```

Element name	Type
<pap>	Contains the push message elements. <b>Attributes:</b> <ul style="list-style-type: none"> <li>• <code>product-name="application_name"</code> Identifies the name of the PAP push initiator.</li> </ul>

Element name	Type
<push-message>	<p>Assigns a unique ID to a push message and defines a notification address and delivery parameters.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"> <li> <p><code>push-id="unique_id"</code></p> <p>Specifies a string that uniquely identifies the message. You can use this ID to cancel or check the status of a message.</p> <p>Typically, you should specify a URL with a value, such as 123@foo.rim.com.</p> </li> <li> <p><code>ppg-notify-requested-to="valid_url"</code></p> <p>Specifies the URL to which the result notification message is sent.</p> </li> <li> <p><code>deliver-after-timestamp="time_stamp"</code></p> <p>Specifies the date and time after which the content must be delivered to the device. Content may not be delivered prior to the specified date and time. The time stamp must be specified using the following notation:</p> <p><code>YYYY-MM-DDT hh:mm:ssZ</code></p> <p><b>Note:</b> The Z parameter indicates that the time stamp uses Co-ordinated Universal Time (UTC).</p> </li> <li> <p><code>deliver-before-timestamp="time_stamp"</code></p> <p>Specifies the date and time by which the content must be delivered to the device. Content that has not been sent by this date is not delivered.</p> <p>The time stamp must be specified using the following notation:</p> <p><code>YYYY-MM-DDT hh:mm:ssZ</code></p> <p><b>Note:</b> The Z parameter indicates that the time stamp uses UTC.</p> </li> </ul>
<address>	<p>Specifies the address of the target device or group of devices. The &lt;push-message&gt; element can contain one or more &lt;address&gt; elements.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"> <li> <p><code>address-value="valid_address"</code></p> <p>Specifies a text string that represents the address of the target device or group of devices. PAP addresses for BlackBerry devices have the following format:</p> <p><code>WAPPUSH=destination_device%3Aport/TYPE=USER@rim.net</code></p> <p>where <i>destination_device</i> represents the email address, PIN, or BlackBerry Group name of the target devices; and <i>port</i> is the device port number that the application uses to listen for push requests. For browser push applications, the port number is 7874.</p> <p><b>Note:</b> All non-alphanumeric characters in the device email address portion of the value, other than "+", "-", ".", and "_", must be represented by their hexadecimal values. The "@" symbol, therefore, is represented by the sequence "%40", and the ":" symbol by "%3A".</p> </li> </ul>

Element name	Type
<quality-of-service>	<p>Specifies the address of the target device.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"> <li>• <code>delivery-method="unconfirmed   confirmed   preferconfirmed"</code> Specifies the delivery reliability level of the message.</li> </ul> <p><b>Note:</b> This attribute does not imply any notification on the part of the BlackBerry MDS Connection Service to the push initiator about the success or failure of delivery. For that, you must use the <code>ppg-notify-requested-to</code> attribute of the &lt;push-message&gt; element.</p> <ul style="list-style-type: none"> <li>• <code>unconfirmed</code>: Equivalent to transport-level acknowledgement. The destination device sends a confirmation acknowledgement message to the BlackBerry MDS Connection when all of the content has been delivered to the device.</li> <li>• <code>confirmed</code>: Equivalent to application-level acknowledgement. The destination device sends a confirmation acknowledgement message to the BlackBerry MDS Connection Service only when the content is viewed in the browser application.</li> <li>• <code>preferconfirmed</code>: If the destination device is running BlackBerry Device Software Version 4.0, it sends a confirmation acknowledgement message to the BlackBerry MDS Connection Service when the content is viewed in the browser application. If the device is running an earlier version of the BlackBerry Device Software, it sends a confirmation acknowledgement message when all of the content has been delivered to the device.</li> </ul>

### PAP cancel message

A PAP cancel message cancels a message that the push initiator previously sent to the BlackBerry MDS Connection Service.

The following is a sample cancel message XML control entity:

```
<pap>
  <cancel-message push-id="123@foo.rim.com">
    <address address-value="WAPPUSH=john.doe%40rim.com%3A7874/TYPE=USER@rim.net" />
    <address address-value="WAPPUSH=jane.doe%40rim.com%3A7874/TYPE=USER@rim.net" />
    <address address-value="WAPPUSH=2100001A%40rim.com%3A7874/TYPE=USER@rim.net" />
  </cancel-message>
</pap>
```

Element name	Type
<pap>	<p>Contains the cancel message elements.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"> <li>• <code>product-name="application_name"</code> Identifies the name of the PAP push initiator.</li> </ul>
<cancel-message>	<p>Instructs the BlackBerry MDS Connection Service to cancel a specified message. If no address is specified, then the BlackBerry MDS Connection Service cancels the push for all intended recipients.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"> <li>• <code>push-id="unique_id"</code> Specifies string used as the push-id value of the message you want to cancel.</li> </ul>

Element name	Type
<address>	<p>Specifies the address of the target device or group of devices. The &lt;cancel-message&gt; element can contain one or more &lt;address&gt; elements.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"> <li>address-value="valid_address"</li> </ul> <p>Specifies a text string that represents the address of the target device or group of devices. PAP addresses for BlackBerry devices have the following format:</p> <p style="padding-left: 40px;">WAPPUSH=destination_device%3Aport/TYPE=USER@rim.net</p> <p>where <i>destination_device</i> represents the email address, PIN, or BlackBerry Group name of the target devices; and <i>port</i> is the device port number that the application uses to listen for push requests. For browser push applications, the port number is 7874.</p> <p><b>Note:</b> All non-alphanumeric characters in the device email address portion of the value, other than "+", "-", ".", and "_", must be represented by their hexadecimal values. The "@" symbol, therefore, is represented by the sequence "%40", and the ":" symbol by "%3A".</p>

### PAP status query message

A PAP status query message requests the status of a message that the push initiator previously sent to the BlackBerry MDS Connection Service.

The following is a sample status query message XML control entity:

```
<pap>
  <statusquery-message push-id="123@foo.rim.com">
    <address address-value="WAPPUSH=john.doe%40rim.com%3A7874/TYPE=USER@rim.net" />
    <address address-value="WAPPUSH=2100001A%40rim.com%3A7874/TYPE=USER@rim.net" />
  </statusquery-message>
</pap>
```

Element name	Type
<pap>	<p>Contains the status query message elements.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"> <li>product-name="application_name"</li> </ul> <p>Identifies the name of the PAP push initiator.</p>
<statusquery-message>	<p>Requests the status of a previously pushed message.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"> <li>push-id="unique_id"</li> </ul> <p>Specifies the string used as the push-id value of the message you want to query.</p>

Element name	Type
<address>	<p>Specifies the address of the target device or group of devices. The &lt;cancel-message&gt; element can contain one or more &lt;address&gt; elements.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"> <li>• <code>address-value="valid_address"</code> Specifies a text string that represents the address of the target device or a group of devices. PAP addresses for BlackBerry devices have the following format:  <code>WAPUSH=destination_device%3Aport/TYPE=USER@rim.net</code>  where <i>destination_device</i> represents the email address, PIN, or BlackBerry Group name of the target devices; and <i>port</i> is the device port number that the application uses to listen for push requests. For browser push applications, the port number is 7874.</li> </ul> <p><b>Note:</b> All non-alphanumeric characters in the device email address portion of the value, other than "+", "-", ".", and "_", must be represented by their hexadecimal values. The "@" symbol, therefore, is represented by the sequence "%40", and the ":" symbol by "%3A".</p>

## BlackBerry MDS Connection Service push attributes: PAP push HTTP header

A single header is used to route the push message through the BlackBerry MDS Connection Service.

- `X-Wap-Application-Id: unique_id`

A header that specifies the application through which the push is being routed. For pushes routed through the BlackBerry MDS Connection Service, this value is "/".

## Browser push HTTP headers

Browser push applications use several RIM proprietary HTTP headers that are included with the pushed content to inform the browser how to manage the content once it arrives on the device.



**Note:** In a PAP push, these headers must be included in the content portion of the pushed multipart message.

The following headers can be included with a pushed message. The BlackBerry MDS Connection Service passes these HTTP headers to the browser.

- `X-Rim-Push-Title: descriptive_title`

Specifies a text string that is used to identify the push application on the Home Screen, in the message list, or in the browser bookmarks.

- `X-Rim-Push-Type: browser-message | browser-content | browser-channel | browser-channel-delete`

Specifies the destination of the pushed content once it is delivered to the browser.

- `browser-message`: Pushes content to the message list, where the message is identified as a browser message.
- `browser-content`: Pushes content directly to the browser's content cache. The next time that the user visits the specified URL, the browser retrieves the updated content from the cache.
- `browser-channel`: Pushes content to the Home Screen, where it is added as an icon. When clicked, the icon opens the browser, which displays the pushed content. If you specify this option, you must also

specify a channel ID using the `X-Rim-Channel-ID` header. If the given channel ID already exists, that channel is updated with the new content.

- `browser-channel-delete`: Deletes a channel. If you specify this option, you must also specify a channel ID using the `X-Rim-Channel-ID` header. The browser deletes the channel with the given channel ID.
- `X-Rim-Push-Channel-ID`: *unique\_id*  
Specifies a string that uniquely identifies a new or existing channel.
- `X-Rim-Push-Read-Icon-URL`: *image\_url*  
Specifies the URL of the icon image that is used to identify the push application.
- `X-Rim-Push-Ribbon-Position`: *position\_number*  
Specifies an integer that defines the position of the channel push icon on the device Home Screen.

## RIM push service implementation

The RIM push service implementation sends the push content as a byte stream to the destination user and port number that is specified in the URL of the push message. RIM push applications use HTTP headers to define the attributes of the application, such as

- the push type (channel, message, or cache)
- a unique identifier for the application
- the location and names of the images used as icons
- the priority of the push
- the reliability level of the push

If the BlackBerry Enterprise Server is configured to do so, all push data is stored in the BlackBerry Enterprise Server databases. Otherwise, it is stored in the RAM of the BlackBerry MDS Connection Service server.

## Writing a RIM push service application

The following is a simple example of a PHP application that pushes a web page to a BlackBerry device. The example relies on an HTML form that supplies the required push information (`index.html`). This is a basic HTML form and will not be discussed in the example, but it is included as part of the code sample. See "Code sample: Creating a browser push application using the RIM push service implementation" on page 81 for more information.

### Create the PHP application

1. Create the URL that the request will be posted to. This can be the email address or BlackBerry PIN of the user to deliver the push to. The BlackBerry device listens to incoming push requests on port 7874.  

```
$path = "/push?DESTINATION=" . $HTTP_POST_VARS['email'] . "&PORT=7874&REQUESTURI="/;
```
2. Open a socket connection to the BlackBerry MDS Connection Server. For this sample, the user supplies the server hostname and port number in the HTML form.

```
$besfs = @fsockopen($HTTP_POST_VARS['besHostname'], $HTTP_POST_VARS['besPort'],
    $errno, $errstr,30);
```

3. Begin sending header properties for the push. For this sample, the user supplies the URL of the page to be pushed, the push title, and the push type in the HTML form.

```
fputs($besfs, "POST $path HTTP/1.0\r\n");

fputs($besfs, "Host: " . $HTTP_POST_VARS['besHostname'] . "\r\n");

fputs($besfs, "Content-Location: " . $HTTP_POST_VARS['pushURL'] . "\r\n");

fputs($besfs, "X-RIM-Push-Title: " . $HTTP_POST_VARS['pushTitle'] . "\r\n");

fputs($besfs, "X-RIM-Push-Type: " . $HTTP_POST_VARS['pushType'] . "\r\n");
```

4. If the Browser Push is a Browser-Channel or Browser-Channel-Delete push, you must specify the "X-RIM-Push-Channel-ID" header property. If a Browser Push is being sent, the URL of the of the read and unread icons can be specified. If no URL is specified, default device icons will be used.

```
if ($HTTP_POST_VARS['pushType'] == "Browser-Channel" || $HTTP_POST_VARS['pushType']
    == "Browser-Channel-Delete")
{
    fputs($besfs, "X-RIM-Push-Channel-ID: " . $HTTP_POST_VARS['pushURL'] . "\r\n");

    //Send the push icon URL if we are using
    //a browser channel push.
    if($HTTP_POST_VARS['pushType'] == "Browser-Channel")
    {
        fputs($besfs, "X-RIM-Push-Read-Icon-URL: " . $HTTP_POST_VARS['readIconURL']
            . "\r\n");
    }
}
```

5. Send the web page to the client. This is not required when sending a Browser-Channel-Delete.

```
$webURL = parse_url($HTTP_POST_VARS['pushURL']);

//Set the port to 80 if not otherwise specified
//in the push URL.
if (empty($webURL['port']))
    $port = 80;
else
    $port = $webURL['port'];

//Open a socket connection to the webserver
//hosting the page to push.
$webfs = @fsockopen($webURL['host'], $port, $errno, $errstr, 30);
fputs($webfs, "GET " . $webURL['path'] . " HTTP/1.1\r\n");
fputs($webfs, "Host: " . $webURL['host'] . "\r\n");
fputs($webfs, "Connection: Close\r\n\r\n");

//Read and discard the first line from the
//webserver (should be HTTP/1.1 200 OK).
fgets($webfs);
```



```
//Read in the page to push from the web server
//and write it out to the MDS server.
while (!feof($webfs))
{
    fputs($besfs, fgets($webfs));
}
//Close the connection to the web server.
fclose($webfs);
```

6. If the push is a Browser-Channel-Delete, you must specify two additional header properties: Content-length and Connection: Close. If you are performing a Browser-Channel, Browser-Message, or Browser-Content Push, the web server provides these properties.

```
if ($HTTP_POST_VARS['pushType'] == "Browser-Channel-Delete")
{
    //No content to send when performing a
    //Browser Channel Delete Push.
    fputs($besfs, "Content-length: 0\r\n");
    fputs($besfs, "Connection: Close\r\n\r\n");
}
```

7. Send the output from the BlackBerry MDS Connection Service to the client. A response code of 200 indicates a successful push.

```
$buf = "";
//Capture the BlackBerry MDS Connection Service's reply to the push.
while (!feof($besfs))
    $buf .= fgets($besfs,128);
```

8. Close the connection.

```
fclose($besfs);
//Display the BlackBerry MDS Connection Service's reply.
echo "Response from MDS: <br>";
echo $buf;
```

## Set up and test the example

To run this push application, simply place the .php file and the HTML form on a PHP-enabled web server and open it in your browser.

In the BlackBerry Device Simulator, the icon that is specified in the unreadIconUrl property appears on the device Home screen. To view the web page that is specified in the pushUrlString property, click the icon.

## Code sample: Creating a browser push application using the RIM push service implementation

This section includes the complete samples for the following files:

- phpRIMPush.php: Pushes the content to the BlackBerry device.
- index.html: Collects the parameters for phpRimPush.php

---

Example: phpRIMPush.php

```
<html>
```

```

<title>Push Results</title>
<body>
<?

//Ensure all required variables have been filled in.
if (!@empty($HTTP_POST_VARS['besHostname']) || !@empty($HTTP_POST_VARS['email'])
    || !@empty($HTTP_POST_VARS['pushURL']) || !@empty($HTTP_POST_VARS['pushType'])
    || !@empty($HTTP_POST_VARS['pushTitle']))
{

    //Ensure that the MDS Push port is numeric.
    if(!is_numeric($HTTP_POST_VARS['besPort']))
    {
        echo "Error! MDS Push Port must be numeric!";
    }
    else
    {

        //Create the push URL.
        $path = "/push?DESTINATION=" . $HTTP_POST_VARS['email'] . "&PORT=7874&REQUESTURI=/";

        //Open a socket connection to the MDS Server.
        if ($besfs = @fsockopen($HTTP_POST_VARS['besHostname'], $HTTP_POST_VARS['besPort'],
            $errno, $errstr, 30))
        {
            fputs($besfs, "POST $path HTTP/1.0\r\n");
            fputs($besfs, "Host: " . $HTTP_POST_VARS['besHostname'] . "\r\n");
            fputs($besfs, "Content-Location: " . $HTTP_POST_VARS['pushURL'] . "\r\n");
            fputs($besfs, "X-RIM-Push-Title: " . $HTTP_POST_VARS['pushTitle'] . "\r\n");
            fputs($besfs, "X-RIM-Push-Type: " . $HTTP_POST_VARS['pushType'] . "\r\n");

            //Send the push URL if we are using a browser channel or browser channel delete.
            if ($HTTP_POST_VARS['pushType'] == "Browser-Channel" ||
                $HTTP_POST_VARS['pushType'] == "Browser-Channel-Delete")
            {
                fputs($besfs, "X-RIM-Push-Channel-ID: " . $HTTP_POST_VARS['pushURL'] .
                    "\r\n");

                //Send the push icon URLs if we are using a browser channel push.
                if($HTTP_POST_VARS['pushType'] == "Browser-Channel")
                {
                    fputs($besfs, "X-RIM-Push-UnRead-Icon-URL: " .
                        $HTTP_POST_VARS['unreadIconURL'] . "\r\n");
                    fputs($besfs, "X-RIM-Push-Read-Icon-URL: " .
                        $HTTP_POST_VARS['readIconURL'] . "\r\n");
                }
            }

            $keepGoing = true;

            // No need to connect to the pushed page for a delete.
            if ($HTTP_POST_VARS['pushType'] != "Browser-Channel-Delete")
            {

                $webURL = parse_url($HTTP_POST_VARS['pushURL']);

                //Set the port to 80 if not otherwise specified in the push URL.
                if (empty($webURL['port']))

```

```

        $port = 80;
    else
        $port = $webURL['port'];

    //Open a socket connection to the web server hosting the page to push.
    if ($webfs = @fsockopen($webURL['host'], $port, $errno, $errstr, 30))
    {
        fputs($webfs, "GET " . $webURL['path'] . " HTTP/1.1\r\n");
        fputs($webfs, "Host: " . $webURL['host'] . "\r\n");
        fputs($webfs, "Connection: Close\r\n\r\n");

        //Read and discard the first line from the web server (should be HTTP/1.1
        //200 OK).
        fgets($webfs);

        //Read in the page to push from the web server and write it out to the
        //BlackBerry MDS Connection Service.
        while (!feof($webfs))
        {
            fputs($besfs, fgets($webfs));
        }

        //Close the connection to the web server.
        fclose($webfs);
    }
    else
    {
        echo "Failed to connect to " . $HTTP_POST_VARS['pushURL'] . "<br>";
        echo "Error: " . $errno . " " . $errstr;

        $keepGoing = false;
    }
}
else
{
    //No content to send when performing a Browser Channel Delete Push.
    fputs($besfs, "Content-length: 0\r\n");
    fputs($besfs, "Connection: Close\r\n\r\n");
}

//If we didn't fail connecting to the web server, lets keep going!
if ($keepGoing)
{
    $buf = "";

    //Capture the BlackBerry MDS Connection Service's reply to the push.
    while (!feof($besfs))
        $buf .= fgets($besfs, 128);
    fclose($besfs);

    //Display the BlackBerry MDS Connection Service's reply.
    echo "Response from MDS: <br>";
    echo $buf;
}
else
{
    //Close the connection to the BlackBerry MDS Connection Service.
    fclose($besfs);
}

```

```

        }
    }
    else
    {
        echo "Failed to connect to " . $HTTP_POST_VARS['besHostname'] . " using port "
            . $HTTP_POST_VARS['besPort'] . "<br>";
        echo "Error: " . $errno . " " . $errstr;
    }
}
}
else
{
    echo "Error! Required fields were empty. Please try again.";
}

?>
</body>
</html>

```

---

**Example: index.html**

```

<html>
<title>Test Push Application</title>
<body>
<form name="phpPush" method="post" action="phpPush.php">
<h2>Sample php Browser Push</h2>
<table border="0">
    <tr>
        <td>BlackBerry Enterprise Server Name/IP Address:</td>
        <td><input type="text" name="besHostname" size="27"></td>
    </tr>
    <tr>
        <td>BlackBerry MDS Connection Service Push Port:</td>
        <td><input type="text" name="besPort" size="50"></td>
    </tr>
    <tr>
        <td>Push Recipient's Email Address:</td>
        <td><input type="text" name="email" size="50"></td>
    </tr>
    <tr>
        <td>Push URL (file to be pushed):</td>
        <td><input type="text" name="pushURL" size="50" value="http://localhost/phpPush/
            testpage/sample.html"></td>
    </tr>
    <tr>
        <td>Push Type:</td>
        <td><select name="pushType">
            <option>Browser-Channel</option>
            <option>Browser-Message</option>
            <option>Browser-Content</option>
            <option>Browser-Channel-Delete</option>
        </select></td>
    </tr>
    <tr>
        <td>Push Title:</td>
        <td><input type="text" name="pushTitle" size="50" value="Test Push"></td>
    </tr>

```

```

</tr>
<tr>
  <td>Unread Icon URL (optional):</td>
  <td><input type="text" name="unreadIconURL" size="50" value="http://localhost/
    phpPush/testpage/smile_unread.png"></td>
</tr>
<tr>
  <td>Read Icon URL (optional):</td>
  <td><input type="text" name="readIconURL" size="50" value="http://localhost/
    phpPush/testpage/smile.png"></td>
</tr>
<tr>
  <td colspan="2">&nbsp;</td>
</tr>
<tr>
  <td></td>
  <td align="right"><input type="submit" name="submit" value="Send Push!"></td>
</tr>
</table>
</form>
</body>
</html>

```

---

## PAP push service implementation

To push data to devices using PAP, send an HTTP POST request to the BlackBerry MDS Connection Service using the following format

`/push?DESTINATION=destination&PORT=port&REQUESTURI=/pap`

where

- *destination* is the URL of the BlackBerry MDS Connection Service
- *port* is the web server listen port. The default port number is 8080 for the BlackBerry MDS Simulator and IBM Lotus Domino, and 8300 for Microsoft Exchange

The request is a MIME multipart message, which consists of the following items:

- an XML document specifying the control entity
- the push content

For example, the control entity might contain information for the device address, message ID, and delivery time stamps.

See the *Push Access Protocol (WAP-247-PAP-20010429-a)* specification for more information about writing server-side push applications using PAP. See the PAP 2.0 DTD for information about the WAP Push DTDs.

## Writing a PAP push service application

This section demonstrates how to write a push application that uses the PAP push service implementation to create an icon on the device Home Screen.

See "Code sample: Creating a browser push application using the PAP push service implementation" on page 90 for the complete code sample, as well as complete samples of the properties file and all the XML control entities used for this example.

## Write a push method

1. Define a method that accepts the required parameters for a channel push.

```
public static void pushPage(String mdsHostName, int mdsPort, String email,
    String pushUrlString, String pushType, String pushTitle,
    String unreadIconUrl, String readIconUrl,
    String pushReliability, String notifyUrl,
    String PushId) {
```

2. Define variables for connections to the BlackBerry MDS Connection Service.

```
    HttpURLConnection mdsConn;
    URL mdsUrl;
```

3. Create the connection to the BlackBerry MDS Connection Service by invoking the `openConnection` method on the URL. The push listener thread on the device listens on port 7874.

```
    mdsUrl = new URL("http", mdsHostName, mdsPort, "/pap");
    mdsConn = (HttpURLConnection)mdsUrl.openConnection();
```

4. Set additional header properties for the push.

```
    try {
        String protocol = "http";
        if (useHttps == true) {
            protocol += "s";
        }
        mdsUrl = new URL(protocol, mdsHostName, mdsPort, "/pap");
        mdsConn = (HttpURLConnection)mdsUrl.openConnection();
        if ((user != null) && (password != null)) {
            String authString = user + ":" + password;
            mdsConn.setRequestProperty("Authorization", "Basic " + new
                BASE64Encoder().encode(authString.getBytes()));
        }
        String boundary = "";
        if ((command == 'p') || (command == 'r')) {
            boundary = "asdlfkjiurwghasf";
            mdsConn.setRequestProperty("Content-Type", "multipart/related;
                type=\"application/xml\"; boundary=" + boundary);
            mdsConn.setRequestProperty("X-Wap-Application-Id", "/");
            mdsConn.setRequestProperty("X-Rim-Push-Dest-Port", "7874");
        }
        else {
            mdsConn.setRequestProperty("Content-Type", "application/xml");
        }

        mdsConn.setRequestProperty("Content-Location", pushUrlString);
        mdsConn.setRequestProperty("X-RIM-Push-Title", pushTitle);
        mdsConn.setRequestProperty("X-RIM-Push-Type", pushType);
        if (pushType.equals(CHANNEL) || pushType.equals(CHANNEL_DELETE)) {
            mdsConn.setRequestProperty("X-RIM-Push-Channel-ID", pushUrlString);
            if (pushType.equals(CHANNEL)) {
                mdsConn.setRequestProperty("X-RIM-Push-UnRead-Icon-URL", unreadIconUrl);
                mdsConn.setRequestProperty("X-RIM-Push-Read-Icon-URL", readIconUrl);
            }
        }
    }
```

```

    }
}

try {
    mdsConn.setRequestMethod("POST");
} catch (ProtocolException e) {
    throw new RuntimeException("problems setting request method: " +
        e.getMessage());
}

mdsConn.setAllowUserInteraction(false);
mdsConn.setDoInput(true);

```

5. Open the connection to the BlackBerry MDS Connection Service.

```

mdsConn.connect();
...

```

6. Retrieve the response code that the BlackBerry MDS Connection Service returns . Display a status message.

```

int rescode = mdsConn.getResponseCode();
if (rescode != HttpURLConnection.HTTP_OK) {
    throw new RuntimeException("Cannot push data, received bad response code
        from Mobile Data Service: " + rescode);
}
System.out.println("Pushed page to the device");

```

## Create a method to retrieve the content

1. Define a method that accepts the parameters that are required to define the push content.
 

```
private static String[] getContent(String pushUrlString, String pushType, String
pushTitle, String unreadIconUrl, String readIconUrl) {
```
2. Define variables for connections to two URLs: one to the server with the content and one to the BlackBerry MDS Connection Service.

```

HttpURLConnection pushConn = null;
URL pushUrl;

```

3. Invoke the openConnection method on the URL to connect to the content server.

```

StringBuffer contentHeaders = new StringBuffer();
try {
    pushUrl = new URL(pushUrlString);
} catch (MalformedURLException e) {
    throw new RuntimeException("invalid push URL: " + e.getMessage());
}
pushConn = (HttpURLConnection)pushUrl.openConnection();

```

4. Set properties for the HTTP GET request to the content server.

```

pushConn.setAllowUserInteraction(false);
pushConn.setDoInput(true);
pushConn.setDoOutput(false);
pushConn.setRequestMethod("GET");

```

5. Open the connection to the content server.

```

pushConn.connect();

```

- Set properties for the HTTP POST request to the BlackBerry MDS Connection Service. Copy the header properties from the GET request. Set additional properties for the push.

```
String name, value;
for (int i = 0; true; i++) {
    name = pushConn.getHeaderFieldKey(i);
    value = pushConn.getHeaderField(i);
    if ((name == null) && (value == null)) { break; }
    if ((name == null) || (value == null)) { continue; }
    if (name.equals("X-RIM-Push-Type")) { continue; }
    if (name.equals("Transfer-Encoding")) { continue; }
    contentHeaders.append("\r\n").append(name).append(": ").append(value);
}
content[0] = contentHeaders.toString();
```

- Read content from the content server connection. Write it to the BlackBerry MDS Connection Service connection.

```
InputStream ins = pushConn.getInputStream();
ByteArrayOutputStream bouts = new ByteArrayOutputStream();
copyStreams(ins, bouts);
ins.close();
content[1] = new String(bouts.toByteArray());

return content;
```



Tip: This step is optional for applications that use the channel or message push methods. If you do not push content to the BlackBerry device, the browser retrieves the content when the user requests it.

If you are pushing content to many users, copy the content to a temporary file to avoid multiple requests to the content server.

## Create a method that reads a property file and pushes the page

- Define the method that accepts an array of arguments that includes the PAP command and PushID.

```
public static void main(String[] args) {
```

- Configure the method to send a different XML file based on the PAP command issued.

```
char command = args[0].charAt(0);
String papFilename;
switch (command)
{
    case 'p':
        papFilename = "pap_push.txt";
        break;
    case 's':
        papFilename = "pap_status.txt";
        break;
    case 'c':
        papFilename = "pap_cancel.txt";
        break;
    case 'r':
        papFilename = "pap_replace.txt";
        break;
    default:
        System.out.println("invalid command");
        return;
}
```



```
}
```

### 3. Read values from a property file.

```
Properties prop = new Properties();
try {
    prop.load(new FileInputStream(PROPERTIES_FILE));
} catch (FileNotFoundException fnfe) {
    throw new RuntimeException("property file not found: " + fnfe.getMessage());
} catch (IOException ioe) {
    throw new RuntimeException("problems reading property file: " +
        ioe.getMessage());
}
```

```
String mdsHostName = prop.getProperty("mdsHostName").trim();
int mdsPort = (new Integer(prop.getProperty("mdsPort").trim())).intValue();
String pushUrlString = prop.getProperty("pushUrlString").trim();
String pushType = prop.getProperty("pushType", CHANNEL).trim();
String pushTitle = prop.getProperty("pushTitle", "Push Page").trim();
String unreadIconUrl = prop.getProperty("unreadIconUrl", "").trim();
String readIconUrl = prop.getProperty("readIconUrl", "").trim();
String user = prop.getProperty("user");
String password = prop.getProperty("password");
```

```
boolean useHttps = false;
try {
    useHttps = Boolean.valueOf(prop.getProperty("useHttps")).booleanValue();
} catch (Exception e) {}

if (useHttps == true) {
    System.setProperty("javax.net.ssl.trustStore",
        "C:\\p4\\main\\IPProxyProject\\webserver\\webserver.keystore");
    System.setProperty("javax.net.ssl.trustStorePassword", "password");
}
```

```
String pushId = args[1];
```

```
String replaceId = "";
if (command == 'r') {
    System.out.print("Enter push id to replace: ");
    replaceId = (new BufferedReader(new
        InputStreamReader(System.in))).readLine().trim();
}
```

### 4. To retrieve the content, invoke the `getContent` method that you created in "Create a method to retrieve the content" on page 87.

```
String[] content = null;
if ((command == 'p') || (command == 'r')) {
    content = getContent(pushUrlString, pushType, pushTitle, unreadIconUrl,
        readIconUrl);
}
```

### 5. Invoke the `pushPage` method that you created in "Write a push method" on page 86.

```
pushPage(command, pushId, replaceId, papFilename, mdsHostName, mdsPort, pushType,
    user, password, useHttps, content);
```

## Set up and test the example

1. Create the XML portion of the PAP push messages. You need to create an XML file for each of the following PAP push commands:

- push content (see "pap\_push.txt" on page 97 for more information).
- replace content (see "pap\_replace.txt" on page 97 for more information).
- push cancellation (see "pap\_cancel.txt" on page 98 for more information).
- query push status (see "pap\_status.txt" on page 98 for more information).



**Note:** In the sample pap\_push.txt and pap\_replace.txt files, you will need to change all <address> elements to reflect the desired recipient addresses.

Recipient addresses conform to WAP-PAP 2.0 addressing standards and have the following format:

```
WAPPUSH=<email_or_PIN>%3A<port_number>/TYPE=USER@rim.net
```

where <email\_or\_PIN> is the email address or the PIN of the client and <port\_number> is the client port number to which the message is pushed.

All non-alphanumeric characters other than the plus sign (+), the minus sign (-), the period (.), and the underscore (\_) in <email\_or\_PIN> must be represented by %<n>, where <n> is the two-digit hexadecimal number representing the character (for example, %3A represents the colon (:) symbol, while %40 represents the at symbol (@)).

For example, for a push to a BlackBerry device with an email address of john.doe@acme.com to the port 4567, the PAP message specifies the address:

```
WAPPUSH=john.doe%40acme.com%3A4567/TYP=USER@rim.net
```

2. Deploy the web content and the push icons that you want to push to devices on a web application server.
3. Place the BrowserChannelPush.java, pappush\_browserpush.properties, pap\_push.txt, pap\_replace.txt, pap\_status.txt, and pap\_cancel.txt files into a com\rim\docs\samples\browserpushdemo folder that is in your CLASSPATH.
4. Edit the pap\_browserpush.properties file to specify the location and names of the files to push to the device.
5. Start the BlackBerry MDS Simulator and the BlackBerry Device Simulator.
6. Turn on push functionality on the BlackBerry Device Simulator.
7. Compile and run the Java file.

In the BlackBerry Device Simulator, the icon that is specified in the unreadIconUr1 property appears on the device Home screen. Click the icon to view the web page that is specified in the pushUr1String property.

## Delete the channel

If necessary, delete the channel.

1. Edit the pappush\_browserpush.properties file.
2. Change the pushType parameter to pushType = Browser-Channel-Delete.
3. Compile and run BrowserChannelPush.java again.

## Code sample: Creating a browser push application using the PAP push service implementation

This section includes the complete samples for the following files:

- BrowserPAPPushDemo.java: Pushes the content to the BlackBerry device.
- pap\_browserpush.properties file: A text file that contains the push type, the URL of the push content and icon, and the BlackBerry MDS Connection Service port.
- pap\_push.txt: The XML control entity used to send content to the browser or browser channel.
- pap\_replace.txt: The XML control entity used to replace an existing channel.
- pap\_cancel.txt: The XML control entity used to cancel a push request.
- pap\_status.txt: The XML control entity used for a status query request.

---

### Example: BrowserPAPPushDemo.java

```
/*
 * BrowserPushDemo.java
 * Copyright (C) 2004 Reasearch In Motion Limited. All rights reserved.
 * PAP Push properties file
 */
// package com.rim.samples.server.browserpushdemo;

import java.io.*;
import java.net.*;
import java.text.*;
import java.util.*;
import javax.net.ssl.*;
import sun.misc.BASE64Encoder;

/**
 * Application which pushes a specified web page to a specified device.
 */
public class BrowserPushDemo {

    public static final String CHANNEL = "Browser-Channel";
    public static final String MESSAGE = "Browser-Message";
    public static final String CONTENT = "Browser-Content";
    public static final String CHANNEL_DELETE = "Browser-Channel-Delete";
    private static final String PROPERTIES_FILE = "browserpush.properties";

    public BrowserPushDemo() { }

    /**
     * Main method which reads the property file and performs the push.
     */

    public static void main(String[] args) {

        // Set cases for each potential pap command.
        char command = args[0].charAt(0);
        String papFilename;
        switch (command)
```

```

{
    case 'p':
        papFilename = "pap_push.txt";
        break;
    case 's':
        papFilename = "pap_status.txt";
        break;
    case 'c':
        papFilename = "pap_cancel.txt";
        break;
    case 'r':
        papFilename = "pap_replace.txt";
        break;
    default:
        System.out.println("invalid command");
        return;
}

// Load the properties file.
Properties prop = new Properties();
try {
    prop.load(new FileInputStream(PROPERTIES_FILE));
} catch (FileNotFoundException fnfe) {
    throw new RuntimeException("property file not found: " + fnfe.getMessage());
} catch (IOException ioe) {
    throw new RuntimeException("problems reading property file: " + ioe.getMessage());
}

// Hostname and Port of the BlackBerry MDS Connection Service that will perform the
// push.
String mdsHostName = prop.getProperty("mdsHostName").trim();
int mdsPort = (new Integer(prop.getProperty("mdsPort").trim())).intValue();

// The URL of the page to push.
String pushUrlString = prop.getProperty("pushUrlString").trim();
String pushType = prop.getProperty("pushType", CHANNEL).trim();
String pushTitle = prop.getProperty("pushTitle", "Push Page").trim();

// URLs of the icons to use for Browser-Channel push type.
String unreadIconUrl = prop.getProperty("unreadIconUrl", "").trim();
String readIconUrl = prop.getProperty("readIconUrl", "").trim();

//Authentication properties.
String user = prop.getProperty("user");
String password = prop.getProperty("password");

// Push security?
boolean useHttps = false;
try {
    useHttps = Boolean.valueOf(prop.getProperty("useHttps")).booleanValue();
} catch (Exception e) { }

if (useHttps == true) {
    System.setProperty("javax.net.ssl.trustStore",
        "C:\\p4\\main\\IPProxyProject\\webserver\\webserver.keystore");
    System.setProperty("javax.net.ssl.trustStorePassword", "password");
}

```

```

String pushId = args[1];

String replaceId = "";
if (command == 'r') {
    System.out.print("Enter push id to replace: ");
    replaceId = (new BufferedReader(new
        InputStreamReader(System.in))).readLine().trim();
}

String[] content = null;
if ((command == 'p') || (command == 'r')) {
    content = getContent(pushUrlString, pushType, pushTitle, unreadIconUrl,
        readIconUrl);
}
// Push the page to the device.
int iterations = Integer.parseInt(prop.getProperty("iterations", "1"));
int delay = Integer.parseInt(prop.getProperty("delay", "5"));
String pushIdBase = pushId;
for (int i = 0 ; i < iterations; i++) {
    if (i > 0) {
        Thread.sleep(delay);
    }
    if (iterations > 1) {
        pushId = pushIdBase + "_" + i;
    }
    pushPage(command, pushId, replaceId, papFilename, mdsHostName, mdsPort, pushType,
        user, password, useHttps, content);
}
}

/*
 * A method used to set the push connection and pass push content to
 * the BlackBerry MDS Connection Service.
 */
private static String[] getContent(String pushUrlString, String pushType, String
    pushTitle, String unreadIconUrl, String readIconUrl) {
    String[] content = new String[2];
    HttpURLConnection pushConn = null;
    URL pushUrl;
    StringBuffer contentHeaders = new StringBuffer();
    try {
        pushUrl = new URL(pushUrlString);
    } catch (MalformedURLException e) {
        throw new RuntimeException("invalid push URL: " + e.getMessage());
    }
    pushConn = (HttpURLConnection)pushUrl.openConnection();
    pushConn.setAllowUserInteraction(false);
    pushConn.setDoInput(true);
    pushConn.setDoOutput(false);
    pushConn.setRequestMethod("GET");

    pushConn.connect();

    contentHeaders.append("Content-Location: ").append(pushUrlString);
    contentHeaders.append("\r\nX-RIM-Push-Title: ").append(pushTitle);
    contentHeaders.append("\r\nX-RIM-Push-Type: ").append(pushType);

    if (pushType.equals(CHANNEL) || pushType.equals(CHANNEL_DELETE)) {

```

```

        contentHeaders.append("\r\nX-RIM-Push-Channel-ID: ").append(pushUrlString);
        if (pushType.equals(CHANNEL)) {
            contentHeaders.append("\r\nX-RIM-Push-UnRead-Icon-URL: ").append(unreadIconUrl);
            contentHeaders.append("\r\nX-RIM-Push-Read-Icon-URL: ").append(readIconUrl);
        }
    }

    /*
     * Read the header properties from the push connection and write
     * them to the BlackBerry MDS Connection Service connection.
     */
    String name, value;
    for (int i = 0; true; i++) {
        name = pushConn.getHeaderFieldKey(i);
        value = pushConn.getHeaderField(i);
        if ((name == null) && (value == null)) { break; }
        if ((name == null) || (value == null)) { continue; }
        if (name.equals("X-RIM-Push-Type")) { continue; }
        if (name.equals("Transfer-Encoding")) { continue; }
        contentHeaders.append("\r\n").append(name).append(": ").append(value);
    }
    content[0] = contentHeaders.toString();

    /*
     * Read content from the push connection and write them to the
     * BlackBerry MDS Connection Service connection.
     */
    InputStream ins = pushConn.getInputStream();
    ByteArrayOutputStream bouts = new ByteArrayOutputStream();
    copyStreams(ins, bouts);
    ins.close();
    content[1] = new String(bouts.toByteArray());

    return content;
}

public static void pushPage(char command, String pushId, String replaceId, String filename,
    String mdsHostName, int mdsPort, String pushType, String user, String
    password, boolean useHttps, String[] content) {

    HttpURLConnection mdsConn;
    URL mdsUrl;

    try {
        /* Push listener thread on the device listens to port 7874 for pushes from
           the BlackBerry MDS Connection Service. */
        String protocol = "http";
        if (useHttps == true) {
            protocol += "s";
        }
        mdsUrl = new URL(protocol, mdsHostName, mdsPort, "/pap");
        mdsConn = (HttpURLConnection)mdsUrl.openConnection();
        if ((user != null) && (password != null)) {
            String authString = user + ":" + password;
            mdsConn.setRequestProperty("Authorization", "Basic " + new
                BASE64Encoder().encode(authString.getBytes()));
        }
        String boundary = "";

```

```

if ((command == 'p') || (command == 'r')) {
    boundary = "asdlfkjiurwghasf";
    mdsConn.setRequestProperty("Content-Type", "multipart/related;
        type=\"application/xml\"; boundary=\"" + boundary);
    mdsConn.setRequestProperty("X-Wap-Application-Id", "/");
    mdsConn.setRequestProperty("X-Rim-Push-Dest-Port", "7874");
}
else {
    mdsConn.setRequestProperty("Content-Type", "application/xml");
}

mdsConn.setRequestProperty("Content-Location", pushUrlString);
mdsConn.setRequestProperty("X-RIM-Push-Title", pushTitle);
mdsConn.setRequestProperty("X-RIM-Push-Type", pushType);
if (pushType.equals(CHANNEL) || pushType.equals(CHANNEL_DELETE)) {
    mdsConn.setRequestProperty("X-RIM-Push-Channel-ID", pushUrlString);
    if (pushType.equals(CHANNEL)) {
        mdsConn.setRequestProperty("X-RIM-Push-UnRead-Icon-URL", unreadIconUrl);
        mdsConn.setRequestProperty("X-RIM-Push-Read-Icon-URL", readIconUrl);
    }
}

try {
    mdsConn.setRequestMethod("POST");
} catch (ProtocolException e) {
    throw new RuntimeException("problems setting request method: " +
        e.getMessage());
}

mdsConn.setAllowUserInteraction(false);
mdsConn.setDoInput(true);
if (!pushType.equals(CHANNEL_DELETE)) {
    mdsConn.setDoOutput(true);
    OutputStream outs = mdsConn.getOutputStream();
    InputStream ins = new BufferedInputStream(new FileInputStream(filename));
    ByteArrayOutputStream bouts = new ByteArrayOutputStream();
    copyStreams(ins, bouts);
    String output = new String(bouts.toByteArray());
    output = output.replaceAll("\\$\\(pushid\\)", pushId);
    if ((command == 'p') || (command == 'r')) {
        output = output.replaceAll("\\$\\(boundary\\)", boundary);
        output = output.replaceAll("\\$\\(headers\\)", content[0]);
        output = output.replaceAll("\\$\\(content\\)", content[1]);
    }
    if (command == 'r') {
        output = output.replaceAll("\\$\\(replaceid\\)", replaceId);
    }
    output = output.replaceAll("\r\n", "EOL");
    output = output.replaceAll("\n", "EOL");
    output = output.replaceAll("EOL", "\r\n");
    copyStreams(new ByteArrayInputStream(output.getBytes()), outs);
}

mdsConn.connect();

// Output headers returned from mdsConn.
String name, value;
for (int i = 0; true; i++) {

```

```

        name = mdsConn.getHeaderFieldKey(i);
        value = mdsConn.getHeaderField(i);
        if ((name == null) && (value == null)) { break; }
    }

    ByteArrayOutputStream output = new ByteArrayOutputStream();
    copyStreams(mdsConn.getInputStream(), output);
    int rescode = mdsConn.getResponseCode();
    if (rescode != HttpURLConnection.HTTP_ACCEPTED) {
        throw new RuntimeException("Unable to send message, received bad response code
            from MDS:" + rescode);
    }
} catch (IOException e) {
    throw new RuntimeException("Unable to send message:" + e.toString());
}
}

/**
 * Reads data from the input stream and copies it to the output stream.
 */

private static void copyStreams(InputStream ins, OutputStream outs) throws IOException {
    int maxRead = 1024;
    byte [] buffer = new byte[1024];
    int bytesRead;

    for(;;) {
        bytesRead = ins.read(buffer);
        if (bytesRead <= 0) {break;}
        outs.write(buffer, 0, bytesRead);
    }
}
}

```

---

#### Example: pap\_browserpush.properties file

```

# pap_browser.properties file

iterations=1
delay=0
debug=true
useHttps = false

# Push from BlackBerry MDS Simulator on the local computer to BlackBerry device simulator.
mdsHostName = localhost

# mdsPort must match WebServer.listen.port set in the BlackBerry MDS Server's
    rimpublish.property file.
mdsPort = 8080

#pushType = Browser-Content
#pushType = Browser-Channel
#pushType = Browser-Channel-Delete
pushType = Browser-Message
pushTitle = Have a Great Day!

```



```
# The actual page (and icons) to push.
pushUrlString = http://localhost/testpage/sample.htm
unreadIconUrl = http://localhost/testpage/smile_unread.png
readIconUrl = http://localhost/testpage/smile.png

# Authentication credentials.
user=pix
password=password
```

---

#### Example: pap\_push.txt

```
--$(boundary)
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"
"http://www.wapforum.org/DTD/pap_2.0.dtd"
[<?wap-pap-ver supported-versions="2.0"?>]>
<pap>
  <push-message push-id="$(pushid)"
    deliver-before-timestamp="2005-09-18T19:50:00Z"
    ppg-notify-requested-to="http://localhost/
      PAPResultNotificationTest.jsp">
    <!-- modify address-value attribute as necessary -->
    <address address-value="WAPUSH=recipient1%40pappush.com%3A7874/TYPE=USER@rim.net"/
      >
    <quality-of-service delivery-method="unconfirmed"/>
  </push-message>
</pap>
--$(boundary)
$(headers)

$(content)
--$(boundary)--
```

---

#### Example: pap\_replace.txt

```
--$(boundary)
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"
"http://www.wapforum.org/DTD/pap_2.0.dtd">
<pap>
  <push-message push-id="$(pushid)"
    replace-push-id="$(replaceid)"
    replace-method="pending-only"
    ppg-notify-requested-to="http://localhost:8080/
      PAPResultNotificationTest.jsp">
    <!-- modify address-value attribute as necessary -->
    <address address-value="WAPUSH=recipient2%40pappush.com/TYPE=USER@rim.net"/>
    <address address-value="WAPUSH=recipient3%40pappush.com/TYPE=USER@rim.net"/>
    <address address-value="WAPUSH=recipient4%40pappush.com/TYPE=USER@rim.net"/>
  </push-message>
</pap>
```

```
        <quality-of-service delivery-method="preferconfirmed"/>
    </push-message>
</pap>
--$(boundary)
$(headers)

$(content)
--$(boundary)--
```

---

#### Example: pap\_cancel.txt

```
<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"
    "http://www.wapforum.org/DTD/pap_2.0.dtd">
<pap>
    <cancel-message push-id="$(pushid)">
    </cancel-message>
</pap>
```

---

#### Example: pap\_status.txt

```
<?xml version="1.0"?>
<!DOCTYPE pap PUBLIC "-//WAPFORUM//DTD PAP 2.0//EN"
    "http://www.wapforum.org/DTD/pap_2.0.dtd">
<pap>
    <statusquery-message push-id="$(pushid)" />
</pap>
```

---

# Testing web pages

## Using the simulators

### Using the simulators

The BlackBerry JDE includes a BlackBerry Device Simulator and a BlackBerry MDS Simulator that let you test your web pages without using a device on an actual wireless network.

#### Start the simulators

1. On the **Start** menu, click **Programs > Research In Motion > BlackBerry MDS Simulator > MDS Simulator**.
2. On the **Start** menu, click **Programs > Research In Motion > BlackBerry Device Simulator > Device Simulator**.

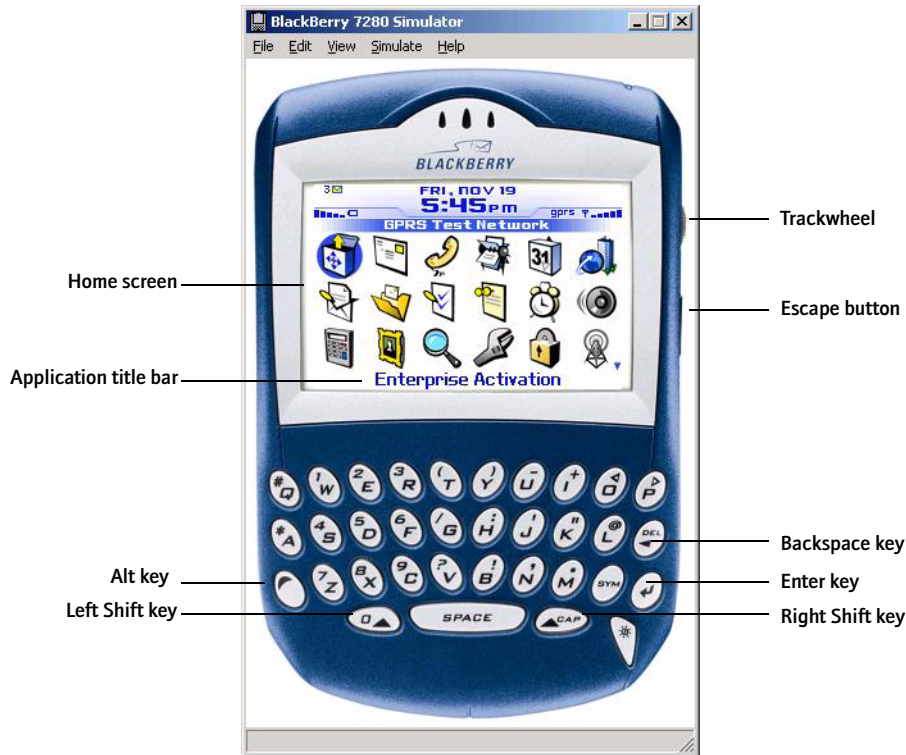
#### Use the BlackBerry Device Simulator

The BlackBerry Device Simulator displays a Home screen with icons for each application.

1. In the BlackBerry Device Simulator, click the **BlackBerry Browser** icon.  
To select the BlackBerry Browser icon, roll the wheel button on your mouse or click the left mouse button. To open the application, click the mouse wheel button or click the right mouse button.
2. To view a web page on an internal network, open the browser menu and click **Go To**. Type the complete URL, such as `http://localhost/tests/sample.wml`.



**Tip:** To open the browser menu, click the mouse wheel button or the right mouse button.



BlackBerry Device Simulator for GPRS

The BlackBerry Device Simulator displays icons on the Home screen for each installed application. When you select an icon, the name of the application appears at the bottom of the screen.

When the BlackBerry Device Simulator window is active, you can roll the wheel button on your mouse to simulate rolling the trackwheel or trackball, and click the wheel button to simulate clicking the trackwheel or trackball. If your mouse does not have a wheel button, click and drag the left mouse button to simulate rolling the trackwheel or trackball, and click the right mouse button to simulate clicking the trackwheel or trackball.

To open an application, roll the trackwheel or trackball to select the appropriate icon, then click the trackwheel or trackball. The application's main screen appears.

Press the keys on your computer keyboard or click the keys on the BlackBerry Device Simulator keyboard to simulate pressing BlackBerry device keys.

See the BlackBerry Device Simulator online help for more information.

# XHTML language reference

XHTML-MP reference  
WAP CSS reference

## XHTML-MP reference



Note: The browser ignores any elements or attributes that are not listed here. Any enclosed text is displayed in normal font, as though the enclosing tags are not present.

### Structural elements

#### <body>

Indicates the start of the page body content.

Attribute	Description
style	Specifies an inline style definition.
title	Describes the contents of the body element.
background	Specifies an image to use as the background.
bgcolor	Specifies the background color of the document.
link	Specifies the color of text used to indicate text links.
text	Specifies the text color used in the document.
xml:lang	Declares the human language used by the element.

#### <frame>

Identifies the content to be displayed in a single pane of a frameset.

While the <frame> element is supported by the browser, this is mainly to support content designed for desktop browsers. You should not include frames in content designed for the BlackBerry device.

Attribute	Description
frameborder	Ignored.
longdesc	Specifies a link to the long description for the frame.
marginheight	Ignored.
marginwidth	Ignored.
name	Specifies a name for the current frame. This attribute is used for targeting content to a particular frame.
noresize	Ignored.
scrolling	Ignored. Because frames are displayed vertically, the browser permits vertical scrolling when necessary.
src	Specifies the URL to be used as the initial content for the frame.

## <frameset>

Presents multiple documents within a single browser window. On a desktop browser, the `<frameset>` element defines the layout of the subwindows, or panes, within the main window; however, when rendered on a BlackBerry device, the layout definition is ignored, and each pane is displayed vertically one after another in the order in which they are encountered.

While the browser supports the `<frame>` element, this is mainly to support content designed for desktop browsers. You should not include frames in content designed for the BlackBerry device.

Attribute	Description
<code>cols</code>	Ignored.
<code>onload</code>	Identifies a script event that is performed when the content of all the frames have been rendered.
<code>rows</code>	Ignored.

## <head>

Contains information about the current document, such as title, keywords that might be useful to search engines, and other data that is not considered document content. User agents do not generally render elements that appear in the head as content. They might, however, make information in the head available to users through other mechanisms.

Attribute	Description
<code>profile</code>	Ignored.

## <html>

The root element of an HTML file.

Attribute	Description
<code>version</code>	Used for grouping only. Deprecated.
<code>lang</code>	Ignored.
<code>xmlns</code>	Ignored.

## <title>

Provides a descriptive title, which appears at the top of the browser screen. This element must be enclosed in the `<head>` tag.

# Text and text formatting elements

## <abbr>

Specifies that the enclosed text is the abbreviated form of a longer word or phrase. Text appears in normal font.

## <acronym>

Specifies that the enclosed text is an acronym. Text appears in normal font.

**<address>**

Specifies that the enclosed text is a mailing address. Text appears in normal font.

**<b>**

Specifies that enclosed text be displayed as bold.

**<big>**

Renders text in a larger font.

**<blockquote>**

Specifies that the enclosed text is part of a quotation. Text appears indented by one character space and appears in normal font.

Attribute	Description
<code>cite</code>	Provides a URL citation to indicate the source of a quotation. The attribute value should be a URL enclosed in quotation marks; the URL appears as an underlined link.
<code>lang</code>	Ignored.

**<br/>**

Starts a new line. To be XHTML-MP-compliant, make sure the element is self-closing.

**<center>**

Horizontally centers the enclosed text.

**<cite>**

Specifies that enclosed text is a citation. Text appears in italic.

**<code>**

Specifies that enclosed text represents code. Text appears in a fixed-width font.

**<dfn>**

Specifies that enclosed text is a term definition. Text appears in italic.

**<div>**

Defines a block of text to which a specific presentation style might be applied.

Attribute	Description
<code>align</code>	Aligns content according to the align attribute. Only horizontal alignment is supported. ( <code>align="left"</code> , <code>align="center"</code> , or <code>align="right"</code> )

**<em>**

Renders the enclosed text in italic.

## <h1> to <h6>

Indicate a heading. Text in heading elements appears in bold. The <h1> element appears in the largest font, with the font size decreasing for each successive heading level.

Attribute	Description
align	Aligns text horizontally. The following values are acceptable: <ul style="list-style-type: none"> <li>align="left"</li> <li>align="right"</li> <li>align="center"</li> </ul>

## <hr/>

Renders a horizontal line. To be XHTML-MP-compliant, make sure the element is self-closing.

## <i>

Renders the enclosed text in italic.

## <kbd>

Represents keyboard input. Text appears in a fixed-width font.

## <p>

Delimits a paragraph of text. Each <p> element starts on a new line.

Attribute	Description
align	Aligns text horizontally. The following values are acceptable: <ul style="list-style-type: none"> <li>align="left"</li> <li>align="right"</li> <li>align="center"</li> </ul>

## <pre>

Formats the enclosed text preserving all spacing and new lines. It does not display text in a fixed-width font.

## <q>

Specifies that enclosed text is a quotation. Text appears in normal font.

Attribute	Description
cite	Provides a URL citation to indicate the source of a quotation. The attribute value should be a URL enclosed in quotation marks; the URL appears as an underlined link.

## <samp>

Designates enclosed text as sample output from programs and scripts, for example. Text appears in a fixed-width font.



**<small>**

Renders text in a smaller font.

**<span>**

Delimits an arbitrary portion of content to which to apply style, display, or event management.

**<strong>**

Specifies that enclosed text be displayed as bold.

**<tt>**

Specifies the enclosed text as teletype. Text appears in a fixed width font.

**<u>**

Renders the enclosed text as underlined.

**<var>**

Indicates a variable name, or a value that the user supplies. Text appears in italic.

Link elements

**<a>**

Indicates an active link.

Attribute	Description
href	Identifies the target of the link. The following link schemes are supported for the href attribute: <ul style="list-style-type: none"><li>• http://</li><li>• mailto:</li><li>• tel:</li><li>• wtai:</li><li>• cti:</li><li>• dc:</li><li>• PIN:</li></ul> Text appears as an underlined link. When users select the link, the browser, phone, or message list opens.
hreflang	Specifies the language of the link reference.
name	Specifies a name for the link. The browser remembers the location as a target for other links.
accesskey	Ignored.
charset	Specifies the character encoding used in the referenced document; the value must be the name of a standard character set.
rel	Specifies the relationship between the current page and the referenced document. For example, "stylesheet".
rev	Provides text that describes a link relationship from the referenced target document to the source document. Not displayed to the user.
style	Specifies an inline style definition.
tabindex	Ignored.
target	Specifies the name of the frame in which to display the content. The _blank, _self, _parent, and _topspecial values are also supported.
title	Provides a descriptive title for the anchor.
type	Specifies the content type of the referenced document; the value is a MIME encoding text, such as "text/plain".

## <link>

Provides an external reference to another document.

Attribute	Description
href	Specifies the target URL of the resource.
rel	Specifies the relationship between the current page and the referenced document. For example, "stylesheet".
target	Specifies the name of the frame in which to display the content.
type	Specifies the MIME type of the target URL.

## List elements

### <dd>

Specifies a definition description. Text appears as a normal paragraph following the <dt> element, with the left margin indented one space to the right.

### <dl>

Specifies a definition list and encloses one or more <dt> tags.

### <dt>

Specifies a definition term. Text appears as a normal paragraph.

### <li>

Specifies a list item. These elements appear with a bullet or number, depending on the enclosing element (<div>, <ul>, or <ol>).

### <ol>

Specifies an ordered (numbered) list. One or more <li> elements enclosed in an <ol> element appear with sequential numbers. The first enclosed <li> element appears on a new line.

### <ul>

Specifies an unordered (bulleted) list. One or more <li> elements enclosed in a <ul> element appear with a bullet. The first enclosed <li> element appears on a new line.

## Basic form elements

### <form>

Specifies a form that gathers information from the user. Users can submit a form by using the <submit> input element. After a submission, the form collects the names and values of enclosed <select>, <input>, and <textarea> elements and submits the query as part of the request (GET) or as post data (POST).

Attribute	Description
action	This attribute provides a URI to which the form is submitted; this attribute is required.
method	The form query is submitted as part of the GET or POST request.
enctype	This attribute is ignored. Form data is encoded with the content type <code>application/x-www-form-urlencoded</code> .
name	Ignored.
style	Ignored.
title	Ignored.

### <input/>

Defines a user input object. The browser renders <input> elements according to the value of the type attribute.

Attribute	Description
type	<p>The following input types are acceptable:</p> <ul style="list-style-type: none"> <li>type="checkbox": The element is rendered using a check box control. Check boxes can occur anywhere in a form element.</li> <li>type="hidden": Hidden elements are not displayed, but they are included when the form is submitted.</li> <li>type="password": The browser displays an asterisk (*) for each character that the user types. The actual value is included in encoded form data when the form is submitted.</li> <li>type="radio": The element is rendered using a radio control (a single selection option list). Radio input elements can appear anywhere in a form element.</li> <li>type="reset": The element appears as a button. Users click the button to reset the form values to its original values. This does not affect other forms on the screen.</li> <li>type="submit": The element appears as a submit button.</li> <li>type="text": The element appears as a text input field.</li> <li>type="img": The associated image that appears is selectable.</li> </ul> <p>The following input types are not supported:</p> <ul style="list-style-type: none"> <li>type="file"</li> <li>type="button"</li> </ul>
maxlength	Sets the maximum number of characters for input elements with type set to password or text.
size	Specifies the width of the text field, in characters.
checked	Supported if type="checkbox". If this attribute is included, the check box appears selected, and the value of the check box is included when the form is submitted.
name	Specifies the name of the check box group; it is required when type="checkbox".
value	Specifies the value to send to the server if a particular check box is selected when the form is submitted; it is required when type="checkbox".

## <label>

Provides a descriptive label for one or more input elements in a form. The label element encloses a text label and one or more `<input>` elements. The text label appears in the browser in normal text.

Attribute	Description
<code>title</code>	The title attribute is ignored.

## <option>

Encloses the text of an option in a select list.

Attribute	Description
<code>selected</code>	Defines the option that is initially selected.

## <select>

Denotes a select list. A select list can be a single-selection or a multiple-selection list. A select list contains one or more option elements.

With single-selection lists, the option with the `selected` attribute set is selected by default; otherwise, the first option in the list is selected by default.

With multiple-selection lists, option elements with the `selected` attribute set are selected by default; otherwise, no options are selected.

Attribute	Description
<code>multiple</code>	Indicates that users are permitted to select more than one option. Rendered as a list of check boxes.

## <textarea>

Denotes a multiline text entry field in a form. It can optionally contain plain text, which is displayed to the user in the text area.

Attribute	Description
<code>rows</code>	Required. Specifies the vertical dimensions of the text area, in number of characters.
<code>cols</code>	Required. Specify the horizontal dimensions of the text area, in number of characters.
<code>name</code>	The name attribute provides plain text to display to the user inside the text area.

## Basic table elements

### <caption>

Provides a description for a table. Enclosed text appears in normal font.

## <table>

Specifies the start of a table; it encloses <thead> and <tbody>, or <caption> and <tr>.

Attribute	Description
border	Specifies the thickness of the border around the outside edges of the table and the inner borders of the table cells.
cellpadding	Specifies the number of pixels of white space to add between cells.
cellspacing	Specifies the number of pixels of white space that is rendered between each adjacent cell.
width	Sets the width of the table, either in pixels or as a percentage of the browser window.

## <td>

Denotes a table cell.

Attribute	Description
colspan	Specifies the number of columns that the cell should span.
rowspan	Specifies the number of rows that the cell should span.
align	Specifies the horizontal alignment of the cell content.
width	Specifies the width of the cell, either in pixels or as a percentage of the table width.

## <th>

Denotes a table heading cell. Text in the cell is bold.

## <tr>

Denotes a table row.

Attribute	Description
align	Specifies the horizontal alignment of the row content.

## Image elements

### <area/>

Defines each area of the image.

Attribute	Description
shape	Defines the shape of the link area. The following values are acceptable: <ul style="list-style-type: none"> <li>shape="circle"</li> <li>shape="rect"</li> <li>shape="polygon"</li> </ul>
coords	This attribute specifies the x and y co-ordinates for the link area. Co-ordinates are comma-delimited.
href	This attribute specifies file name and location of the link.

## <img/>

Defines an image. Monochrome devices support .png and .gif graphics only. With the BlackBerry Browser and Internet Browser configurations, the BlackBerry MDS Data Optimization Service converts .jpeg images to .png format for display on monochrome devices.

Attribute	Description
src	The browser displays the image that is specified by the src attribute. This attribute is required.
alt	Specifies the text that appears when an image is unavailable, provided the browser has been properly configured to display it.
usemap	Specifies the location of the map element.
height	Specifies the height of the image with the unit of measurement. For example, "10pt" or "6px".
width	Specifies the width of the image with the unit of measurement. For example, "10pt" or "6px".
align	Images are aligned horizontally according to the align attribute. The following values are acceptable: <ul style="list-style-type: none"> <li>align="left"</li> <li>align="right"</li> </ul> Vertical alignment is ignored.

## <map>

Creates an image map.

Attribute	Description
name	Identifies the image map. The value must match the corresponding usemap value specified for the image.

## Object elements

### <object>

The browser supports embedded content such as PME (transcoded SVG) content, MIDI files, and other HTML pages. The browser does not support embedding of applets; however, the browser displays embedded content only if it has been properly set to do so by the end user.

### <param>

Defines a parameter for an object.

## Meta information

### <base>

Specifies an absolute URI that acts as the base URI for resolving relative URIs.

Attribute	Description
href	Specifies an absolute URI that acts as the base URI for resolving relative URIs.

## <meta>

This element provides additional information about the document. If the meta element is included, the content attribute is required.

Attribute	Description
http-equiv	<p>Defines the shape of the link area. The following values are acceptable:</p> <ul style="list-style-type: none"> <li><code>refresh</code>: Redirects the browser to the URL specified by the content attribute.</li> <li><code>expires</code>: Specifies that the browser should remove the page from the cache after the number of seconds specified by the content attribute.</li> </ul> <p><b>Note:</b> This tag should be used sparingly for pages intended for wireless browsing because it increases user traffic on the wireless network and can increase the time it takes to display the page.</p> <ul style="list-style-type: none"> <li><code>cache-control</code>: Indicates that cache control parameters have been set for the page, specified by the content attribute.</li> </ul>
content	<p>Required. Sets the meta information for http-equiv. If:</p> <ul style="list-style-type: none"> <li><code>http-equiv="refresh"</code>: Specifies the URL to which the browser is redirected.</li> <li><code>http-equiv="expires"</code>: Specifies the number of seconds after which the page is cleared from the cache. A value of -1 indicates that the page is not cached, so that the browser requests and downloads the page every time it is accessed.</li> <li><code>http-equiv="cache-control"</code>: Specifies how the page is cached. The following values are acceptable: <ul style="list-style-type: none"> <li><code>Public</code>: Page can be stored in public shared caches.</li> <li><code>Private</code>: Page can be stored only in private caches.</li> <li><code>No-cache</code>: Page can not be cached.</li> <li><code>No-store</code>: Page can be cached but not archived.</li> </ul> </li> </ul>

## Script references

### <script>

Defines a script, such as a JavaScript.

Attribute	Description
type	Specifies the MIME type of the script.
language	Deprecated.

## WAP CSS reference

### background

A shorthand property that sets some or all the background properties. Properties can be listed in any order.



**Note:** Do not set a background-attachment value unless you are applying background to the <body> element.

Value	Description
<i>property_list</i>	Specifies a whitespace-separated list of values for the background-attachment, background-color, background-image, and background-repeat properties. For example:  <pre>blockquote {background: red url("images/sand.gif") no-repeat center}</pre> See the descriptions of the background-attachment, background-color, background-image, and background-repeat properties for more information about valid values.

### background-attachment

For background images that are set behind an entire document, as a property applied to the <body> element, the background-attachment property determines whether a background image is fixed and content scrolls across it, or whether the background scrolls with the content.

Value	Description
fixed	Indicates that the content scrolls, but the background image does not.
scroll	Default. Indicates that the background image scrolls with the content.

### background-color

Sets the background color of content and padding. If you set a background color, set the foreground to a contrasting color, so that its content remains visible.

Value	Description
<i>color_value</i>	Any valid color. A valid color can be specified in the following ways: <ul style="list-style-type: none"> <li>an RGB value ("250, 239, 111")</li> <li>hexidecimal notation ("770aff" or "#70f" (equal to "#7700ff"))</li> <li>a valid textual color name ("red")</li> </ul>
transparent	Default. Makes the background invisible, lets the color of the parent element be seen.

### background-image

Sets an image as a background pattern.

Value	Description
URL	Specifies the location of the image to be used for the background pattern.
none	Default. No background pattern is used.



## background-repeat

Defines whether the background image is repeated (tiled) to fill the display.

Value	Description
repeat	Default. The background image is tiled both horizontally and vertically.
repeat-x	The background image is tiled horizontally and is left-aligned.
repeat-y	The background image is tiled vertically and is vertically centered.
no-repeat	The background image is not tiled and is vertically centered and left-aligned.

## border

A shorthand property for setting the width, color, and style of the individual sides of the border. Properties can be listed in any order.

Value	Description
<i>property_list</i>	Specifies a whitespace-separated list of values for the border-color, border-style, and border-width properties. For example:  <pre>table {border: red solid thick}</pre> See the descriptions of the border-color, border-style, and border-width properties for more information about valid values.

## border-bottom, border-left, border-right, border-right

Shorthand properties for setting the color, style, and width of the individual sides of the border. Properties can be listed in any order.

Value	Description
<i>property_list</i>	Specifies a whitespace-separated list of values for the color, style, and width of the specified side of a border. For example:  <pre>table {border-right: red solid thick}</pre> See the descriptions of the border-color, border-style, and border-width properties for more information about valid values.

## border-bottom-color, border-left color, border-right-color, border-top-color

Sets the color of the individual sides of the border.



**Note:** You must set a border-style property to solid to display a border.

Value	Description
<i>color_value</i>	Any valid color. A valid color can be specified in the following ways: <ul style="list-style-type: none"> <li>an RGB value ("250, 239, 111")</li> <li>hexadecimal notation ("770aff" or "#70f", which is equal to "#770ff")</li> <li>a valid textual color name ("red")</li> </ul> The default color is the current foreground color of the element.

## border-bottom-style, border-left-style, border-right-style, border-top-style

Sets the style of the individual sides of the border.

Value	Description
<code>none</code>	Default. The element has no border.
<code>hidden</code>	A hidden border is used. Hidden borders have no width; the browser ignores them.
<code>solid</code>	A solid line is used as a border for the element.

## border-bottom-width, border-left-width, border-right-width, border-top-width

Sets the width of the individual sides of the border.



**Note:** You must set a `border-style` property to `solid` to display a border.

Value	Description
<i>width_value</i>	Specifies the width of the border. The values for these properties can be any integer with the unit of measurement. For example, "10pt" or "6px".  Possible units include <code>em</code> (ems), <code>ex</code> (x-height), <code>px</code> (pixels), <code>in</code> (inches), <code>cm</code> (centimeters), <code>mm</code> (millimeters), <code>pt</code> (points), and <code>pc</code> (picas).
<code>thin</code>	Sets the element border to the browser's interpretation of a thin width.
<code>medium</code>	Default. Sets the element border to the browser's interpretation of a medium width.
<code>thick</code>	Sets the element border to the browser's interpretation of a thick width.
<code>inherit</code>	Indicates that the element inherits the border width of the containing element.

## border-color

A shorthand property for setting the color of all four borders.

Value	Description
<i>color_value</i>	Any valid color. A valid color can be specified in the following ways: <ul style="list-style-type: none"> <li>an RGB value ("250, 239, 111")</li> <li>hexadecimal notation ("770aff" or "#770ff", which is equal to "#7700ff")</li> <li>a valid textual color name ("red")</li> </ul> The default color is the current foreground color of the element.

## border-style

A shorthand property for setting the style of all four borders.

Value	Description
<code>none</code>	Default. The element has no border.
<code>hidden</code>	A hidden border is used. Hidden borders have no width; the browser ignores them.
<code>solid</code>	A solid line is used as a border for the element.

## border-width

A shorthand property for setting the width of all four borders.

Value	Description
<i>width_value</i>	Specifies the width of the border. The value for this property can be any integer with the unit of measurement. For example, "10pt" or "6px".  Possible units include em (ems), ex (x-height), px (pixels), in (inches), cm (centimeters), mm (millimeters), pt (points), and pc (picas).
thin	Sets the element border to the browser's interpretation of a thin width.
medium	Default. Sets the element border to the browser's interpretation of a medium width.
thick	Sets the element border to the browser's interpretation of a thick width.
inherit	Indicates that the element inherits the border-width property of the containing element.

## color

Sets the foreground color. The foreground is the content of the element, typically text, but also, for example, the rule drawn for the `<hr/>` element, and the color of any border (unless you set another color with one of the border properties).

Value	Description
<i>color_value</i>	Any valid color. A valid color can be specified in the following ways: <ul style="list-style-type: none"> <li>an RGB value ("250, 239, 111")</li> <li>hexidecimal notation ("770aff" or "#70f", which is equal to "#7700ff")</li> <li>a valid textual color name ("red")</li> </ul>

## font

A shorthand property that sets the font-family, font-size, font-style, and font-weight for text. Properties can be listed in any order.

Value	Description
<i>property_list</i>	Specifies a whitespace-separated list of values for the font-family, font-size, font-style, and font-weight properties. For example:  <pre>p {font: italic bold "Times New Roman" 10px}</pre> See the descriptions of the font-family, font-size, font-style, and font-weight properties for more information about valid values.

## font-family

Sets a specific or generic font for text.

Value	Description
<i>font_name</i>	A specific font family name, such as "Arial" or "Times New Roman". Font names must be enclosed in quotation marks.
<i>generic_font</i>	A font keyword, such as serif, sans-serif, cursive, and monospace. Generic font keywords should not be enclosed in quotation marks.
inherit	Indicates that the element inherits the font-family property of the containing element.

## font-size

Sets the size of a font, either in absolute or relative terms. Relative values are relative to the font size of the element that contains the current element.

Value	Description
<i>size_value</i>	Any integer with the unit of measurement. For example, "10pt" or "6px". Possible units include em (ems), ex (x-height), px (pixels), in (inches), cm (centimeters), mm (millimeters), pt (points), and pc (picas).
<i>absolute_size</i>	Renders text according to a browser standard. Possible values include xx-small, x-small, small, medium, large, x-large, and xx-large.
<i>relative_size</i>	Renders text at a size relative to the standard font size of the element. Possible values include smaller and larger.
<i>inherit</i>	Indicates that the element inherits the font-size property of the containing element.

## font-style

Sets the font style for text.

Value	Description
<i>normal</i>	Default. No font style is applied.
<i>italic</i>	Uses a font with "italic" in its name or, failing that, one with "oblique" in its name. Italic fonts are also sometimes named cursive or kursive.
<i>oblique</i>	Uses a font with "oblique" in its name or, failing that, one with "italic" in its name. Oblique fonts are also sometimes named slanted or inclined.
<i>inherit</i>	Indicates that the element inherits the font-style property of the containing element.

## font-weight

Sets the thickness of the font.

The browser uses a roman font for font-weight values from 100 to 400 and a bold font for values from 500 to 900.

Value	Description
<i>weight_value</i>	Specifies the absolute weight of the font. Possible values include 100 (lightest), 200, 300, 400, 500, 600, 700, 800, and 900 (heaviest).
<i>normal</i>	Sets the font to standard roman font weight. Equivalent to 400 on the numerical scale.
<i>bold</i>	Sets the font to standard bold font weight. Equivalent to 700 on the numerical scale.
<i>bolder</i>	Increases the font weight by one level on the numerical scale. If the font already had a weight equivalent to 900, it remains 900.
<i>lighter</i>	Decreases the font weight by one level on the numerical scale. If the font already had a weight equivalent to 100, it remains 100.
<i>inherit</i>	Indicates that the element inherits the font-weight property of the containing element.

## height, width

Set the height and width of the content.

Value	Description
<i>measure_value</i>	Specifies the height or width. The value of these properties can be any integer with the unit of measurement. For example, "10pt" or "6px".  Possible units include em (ems), ex (x-height), px (pixels), in (inches), cm (centimeters), mm (millimeters), pt (points), and pc (picas).
auto	Adjusts the height or width of the element to fit the content.
inherit	Indicates that the element inherits the height or width property of the containing element.

## table-layout

Controls the layout of table cells, rows, and columns.

Value	Description
fixed	Specifies that the width of a cell is fixed to a specified width. In this case, the table layout does not depend on the content.  For example, if a column is fixed to a width of 25 pts, any line of text that exceeds this length is broken and continued on a new line.
auto	Specifies that the width of the cell is adjusted to the content. In this case, the table layout depends on the content.  For example, if a line of text is 40 pts in length, the cell is adjusted to a width of 40 pts plus any specified margin width.
inherit	Indicates that the element inherits the table-layout property of the containing element.

## text-align

Sets the horizontal alignment of lines of text.

Value	Description
left	Default. Aligns text to the left margin.
right	Aligns text to the right margin.
center	Centers the text in the display.
inherit	Indicates that the element inherits the text-align property of the containing element.

## text-decoration

Sets underlining and blinking of text.

Value	Description
none	Default. No text decoration is applied.
underline	Underlines the text.
blink	Causes the text to blink.
inherit	Indicates that the element inherits the text-decoration property of the containing element.

## -wap-input-format

Sets an input mask for text input in a form.

To limit the number of characters users can type, specify a single-digit number before the character tag. For example, “3x” requires the user to type a maximum of three symbolic, numeric, or uppercase alphabetic characters.

To allow users to type an unlimited number of characters, specify an asterisk (\*) before the character tag. For example, “\*a” lets the user type any number of symbolic or lowercase alphabetic characters.

To insert a character into the mask, use the syntax \c, replacing the c with the character that you want to insert. This is useful for inserting, for example, a dash in a nine-digit area code.

Value	Description
A	Any symbolic or uppercase alphabetic character (no numbers).
a	Any symbolic or lowercase alphabetic character (no numbers).
N	Any numeric character (no symbols or alphabetic characters).
X	Any symbolic, numeric, or uppercase alphabetic character (not changeable to lowercase).
x	Any symbolic, numeric, or lowercase alphabetic character (not changeable to uppercase).
M	Default. Any symbolic, numeric, or uppercase alphabetic character (changeable to lowercase). For multiple character input, the user input automatically defaults to an uppercase first character.
m	Any symbolic, numeric, or lowercase alphabetic character (changeable to uppercase). For multiple character input, the user input automatically changes to a default lowercase first character.

## -wap-input-required

Requires a user to type text, click a button, or click a menu item on a form.

Value	Description
True	Specifies that the user is required to supply input.
False	Specifies that the user is not required to supply input.

## -wap-marquee-dir

Controls whether content scrolls from left to right (ltr) or from right to left (rtl).



**Note:** This property requires that the selected element also has the display property set to the -wap-marquee value.

Value	Description
ltr	Scrolls content from left to right.
rtl	Scrolls content from right to left.

## -wap-marquee-loop

Controls how many times the marquee effect repeats.



**Note:** This property requires that the selected element also has the display property set to the -wap-marquee value.

Values	Description
<i>iterations</i>	Indicates the number of iterations the marquee effect performs. The value can be any integer. Setting this value to 0 has the same effect as not setting the display property to <code>-wap-marquee</code> .
<i>infinite</i>	The marquee effect loops indefinitely.

## `-wap-marquee-speed`

Controls the speed of the marquee effect.



**Note:** This property requires that the selected element also has the display property set to the `-wap-marquee` value.

Value	Description
<i>slow</i>	Scrolls text across the screen slowly.
<i>normal</i>	Scrolls text across the screen at the standard scroll rate.
<i>fast</i>	Scrolls text across the screen quickly.

## `-wap-marquee-style`

Controls how content scrolls across the screen.



**Note:** This property requires that the selected element also has the display property set to the `-wap-marquee` value.

Value	Description
<i>scroll</i>	The content starts completely off one side of the screen and then scrolls across the screen until it is completely off the other side of the screen, then repeats.
<i>slide</i>	The content starts completely off one side of the screen and then scrolls across the screen. Scrolling stops when the first character reaches the other side of the screen.
<i>alternate</i>	The content starts completely off one side of the screen, and scrolls across the screen until it is completely off the other side of the screen. It does the same thing in the reverse direction, and then repeats.

## Element and CSS property matrix

CSS property	XHTML tag										
	<a>	<b>	<big>	<blink>	<blockquote>	<body>	<button>	<center>	<cite>	<code>	<dd>
background						✓					
background-attachment	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
background-color	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
background-image						✓					
background-repeat						✓					
border							✓				
border-bottom							✓				
border-bottom-color							✓				
border-bottom-style							✓				
border-bottom-width							✓				
border-color							✓				
border-left							✓				
border-left-color							✓				
border-left-style							✓				
border-left-width							✓				
border-right							✓				
border-right-color							✓				
border-right-style							✓				
border-right-width							✓				
border-style							✓				
border-top							✓				
border-top-color							✓				
border-top-style							✓				
border-top-width							✓				
border-width							✓				
color	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
font	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
font-family	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
font-size	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
font-style	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
font-weight	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
height							✓				
table-layout											
text-align	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
text-decoration	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
width							✓				
-wap-input-format											
-wap-input-required											
-wap-marquee-dir											
-wap-marquee-loop											
-wap-marquee-speed											
-wap-marquee-style											

<sup>1</sup> Applies only to <input> elements where type is one of “button”, “submit”, or “reset”.

<sup>2</sup> Applies only to <input> elements where type=“text”.



## Element/CSS property matrix (continued)

CSS property	XHTML tag															
	<marquee>	<menu>	<col>	<optgroup>	<option>	<p>	<pre>	<s>	<samp>	<select>	<small>	<span>	<strike>	<strong>	<sub>	<sup>
background																
background-attachment																
background-color	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
background-image																
background-repeat																
border										✓						
border-bottom										✓	✓					
border-bottom-color										✓	✓					
border-bottom-style										✓	✓					
border-bottom-width										✓	✓					
border-color										✓	✓					
border-left										✓	✓					
border-left-color										✓	✓					
border-left-style										✓	✓					
border-left-width										✓	✓					
border-right										✓	✓					
border-right-color										✓	✓					
border-right-style										✓	✓					
border-right-width										✓	✓					
border-style										✓	✓					
border-top										✓	✓					
border-top-color										✓	✓					
border-top-style										✓	✓					
border-top-width										✓	✓					
border-width										✓	✓					
color	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
font	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
font-family	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
font-size	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
font-style	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
font-weight	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
height																
table-layout												✓				
text-align	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
text-decoration	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
width															✓	
-wap-input-format															✓	
-wap-input-required																✓
-wap-marquee-dir						✓										
-wap-marquee-loop						✓										
-wap-marquee-speed						✓										
-wap-marquee-style						✓										



# WML language reference

WML reference

## WML reference

The following table summarizes browser support for each WML element and its attribute(s). The browser supports WML 1.3.

### Structure elements

#### <access>

If specified, the browser compares the <access> element to the domain and path that are specified in the <access> element to determine whether the user has the proper access to display the page. If not, the browser displays an error message.

Attribute	Description
domain	Specifies the domain used to verify access privileges.
path	Specifies the path used to verify access privileges.

#### <card>

Contains the entire card.

Attribute	Description
title	Displays the card title in the Title area of the screen.
newcontext	If set to true, this attribute clears the variable in store.
ordered	Ignored. All elements in a page are rendered.
onenterforward	Opens the specified URL when the card is accessed using a <go> task.
onenterbackward	Opens the specified URL when the card is accessed using a <prev> task.
ontimer	Opens the specified URL when the timer has expired.

#### <head>

A container element for information on access control that applies to the entire deck (the collection of all cards). See the <access> element.

#### <meta>

Not supported. The browser ignores any <meta> tags.

## <template>

Specifies deck-level <do> or <onevent> items. These items are included in every card for the deck, unless the card has a more specific <do> or <onevent> element that shadows those in the template.

The browser manages template attributes as though they are <onevent> definitions for the corresponding events.

Attribute	Description
onenterforward	Treated as an onenterforward <onevent> element with a <go> action.
onenterbackward	Treated as an onenterbackward <onevent> element with a <go> action.
ontimer	Treated as an ontimer <onevent> element with a <go> action.

## <wml>

Defines a WML deck.

## Text and text formatting elements

### <b>

Renders text in bold in the current font and size, if available. Otherwise, the standard font is used.

### <big>

Renders text in the next larger size of the current font, if available. Nesting of <big> and <small> elements is respected.

### <br>

Starts a new line.

### <p>

Denotes a new paragraph and renders content according to the attributes.

For layout purposes, the <p> and <br /> elements are the same.

Attribute	Description
align	Specifies the position of the contents of the <p> element on screen.
mode	Ignored. Content is always wrapped.

### <em>

Renders text in italic in the current font, if available. Otherwise, the standard font is used.

### <i>

Renders text in italic in the current font and size, if available. Otherwise, the standard font and size is used.

**<pre>**

Not supported.

**<small>**

Renders text in the next smaller size of the current font, if available. The browser supports nesting of big and small elements.

**<strong>**

Renders text in the bold font of the current size, if available. Otherwise, the standard font is used.

**<u>**

Underlines content enclosed in the <u> element in the current font and size, if available. Otherwise, the standard font and size are used.

Link elements

**<a>**

Specifies a link to follow.

Attribute	Description
href	Specifies the target of the link. The browser substitutes any variable references from the WAP context.
title	Ignored.
accesskey	Ignored.

**<anchor>**

When the user moves the cursor over a character or image that is contained in the <anchor> element, users can click the Follow Link menu item to perform the <go>, <prev>, or <refresh> task that is associated with the <anchor> element.

Attribute	Description
domain	Specifies the domain used to verify access privileges.
path	Specifies the path used to verify access privileges.

## Table elements

### <table>

Denotes a new table.

Attribute	Description
<code>title</code>	Ignored.
<code>align</code>	Aligns the text in table columns according to the <code>align</code> value. This attribute is optional. The content of each cell is left-aligned by default.  For example, <code>align="LCR"</code> . Content in the first column is left-justified, content in the second column is centered, and content in the third column is right-aligned.
<code>columns</code>	Required. Specifies the number of columns in the table.

### <tr>

Denotes a new table row.

### <td>

Denotes a new table cell.

## Image elements

### <img>

Defines an image. When images are contained in `<anchor>` or `<a>` elements, users can select the image.

Attribute	Description
<code>alt</code>	Specifies the text that appears when an image is unavailable, provided that the browser is properly configured to display it. When the real image arrives, the real image replaces the <code>alt</code> placeholder.
<code>src</code>	Specifies the location of the image on the server.
<code>localsrc</code>	Unsupported. The browser does not support <code>&lt;localsrc&gt;</code> images.
<code>vspace</code>	Specifies the amount of white space to insert above and below the image.
<code>hspace</code>	Specifies the amount of white space to insert to the left and right of the image.
<code>align</code>	Not supported. Each image is displayed on its own line.
<code>height</code>	Specifies the image height.
<code>width</code>	Specifies the image width.

## Event elements

### <do>

Defines an event trigger. The <do> element can enclose the following task elements: <go>, <prev>, <noop>, or <refresh>.

In the browser, <do> elements that are defined on a card appear both as menu items on the browser menu and as soft keys in a non-scrolling area at the bottom of the screen.

Attribute	Description
type	Required. Specifies the type of <do> element. The <do> element type can be one of accept, help, or prev. A <do> element of type accept appears first on the menu.  The type attribute also matches <do> elements in the deck template, when the name attribute is not defined, for shadowing purposes.
label	Specifies the label used for the associated menu item. If a label attribute is not included, a default name is assigned to the menu item based on type.
name	Matches <do> elements in the deck template for shadowing purposes.
optional	<do> elements with the optional attribute set appear after other <do> elements on the menu.

### <onevent>

Specifies how events are managed.

Attribute	Description
type	Required. Identifies the event to manage; it also matches <do> elements in the deck template for shadowing.  The type attribute can have one of the following values: <ul style="list-style-type: none"><li>• onenterbackward: Occurs when a user navigates backward to a card.</li><li>• onenterforward: Occurs when a user navigates forward to a card.</li><li>• onpick: Occurs when a user selects or clears an option.</li><li>• ontimer: Occurs when the timer, specified by the &lt;timer&gt; element, expires.</li></ul>
id	Sets a unique name for the element.

### <postfield>

Specifies name and value pairs that are included in the HTTP request. <postfield> elements must be enclosed in a <go> element. Variable references in the name and value attributes of each <postfield> element are replaced with appropriate values from the WAP context.

Attribute	Description
name	Required. Specifies the name of the variable to be passed to the server.
value	Required. Specifies the value of the variable to be passed to the server.
id	Sets a unique name for the element.

## Task elements

### <go>

Directs the browser to a specified URI. The browser sets any variables that are specified in `<setvar>` elements within the `<go>` element, and includes any values that are specified in contained `<postfield>` elements.

Attribute	Description
<code>href</code>	Specifies the target URI that the browser goes to.
<code>sendreferer</code>	Specifies whether the URI is sent in the request. Valid values are yes or no.
<code>method</code>	Specifies the request type. The browser supports both the GET and POST methods of sending requests.
<code>enctype</code>	Ignored. The browser uses the default encoding: <code>application/x-www-form-urlencoded</code> .
<code>cache-control</code>	Forces page retrieval from the network instead of the cache. If this attribute is set to <code>no-cache</code> , a flag is set on the request being sent to invalidate this page in the cache, if it is there, before retrieving the page.
<code>accept-charset</code>	Ignored. The browser uses the default character set: UTF-8.

### <noop>

Specifies whether the browser effectively removes any `<do>` or `<onevent>` elements from the current card.

### <prev>

Directs the browser to a specified URI. The inclusion of the `<prev>` element in `<do>` elements affects menu construction. The browser menu always contains at least one item that lets users go back in the navigation history.

If the current card does not contain any `<do>` elements with a `<prev>` task, by default the browser creates a Back menu item. If the current card contains one or more `<do>` elements that contain `<prev>` elements, the browser does not create a separate menu item and relies on the `<do>` elements for that card to provide this behavior instead.

### <refresh>

Refreshes any variables that are specified by the enclosed `<setvar>` element(s). If an event occurs that has a `<refresh>` task, the browser first sets any variables that are specified in `<setvar>` elements that are contained in the `<refresh>` element. It then refreshes the current page using the updated WAP context.



## Input elements

### <fieldset>

Ignored. Enclosed elements are rendered as though the <fieldset> element did not exist.

### <input>

Renders as a text field into which users can type text. If users type a value that is not consistent with the restrictions specified by the attributes in the input element, the browser displays a warning when users try to save the value.

Input boxes appear on a separate line from the surrounding text.

Attribute	Description
name	Specifies the name of the variable to set with any typed value.
type	Specifies the type of information being collected. Valid values include <code>text</code> or <code>password</code> . If type is set to <code>password</code> , the browser displays an asterisk (*) for each character that the user types.
value	Specifies the initial value that is displayed in the input field when the card is first rendered. This value is used only when the variable specified by the <code>name</code> attribute is not set already in the WAP Context. If the variable is set, the current value of that variable is used instead.
format	<p>Specifies a mask. The browser checks that any text the user enters in the input field conforms to the mask that is specified by this attribute. The following character tags define which characters can be typed:</p> <ul style="list-style-type: none"><li>• <code>A</code> – Any symbolic or uppercase alphabetic character (no numbers).</li><li>• <code>a</code> – Any symbolic or lowercase alphabetic character (no numbers).</li><li>• <code>N</code> – Any numeric character (no symbols or alphabetic characters).</li><li>• <code>X</code> – Any symbolic, numeric, or uppercase alphabetic character (not changeable to lowercase).</li><li>• <code>x</code> – Any symbolic, numeric, or lowercase alphabetic character (not changeable to uppercase).</li><li>• <code>M</code> – Default. Any symbolic, numeric, or uppercase alphabetic character (changeable to lowercase). For multiple character input, the user input automatically changes to a default uppercase first character.</li><li>• <code>m</code> – Any symbolic, numeric, or lowercase alphabetic character (changeable to uppercase). For multiple character input, the user input automatically changes to a default lowercase first character.</li></ul> <p>Tips:</p> <ul style="list-style-type: none"><li>• To limit the number of characters users can type, specify a single-digit number before the character tag. For example, "<code>3X</code>" requires the user to type a maximum of three symbolic, numeric, or uppercase alphabetic characters.</li><li>• To let users type an unlimited number of characters, specify an asterisk (*) before the character tag. For example, "<code>*a</code>" lets the user type any number of symbolic or lowercase alphabetic characters.</li><li>• To insert a character into the mask, use the syntax <code>\c</code>, replacing the <code>c</code> with the character that you want to insert. This is useful for inserting, for example, a dash in a nine-digit area code.</li></ul>
emptyok	Lets the user leave the field blank.
size	Specifies the size, in characters, of the input box.
maxlength	Specifies the maximum length of text the user can type.
tabindex	Ignored. Tabbing is not supported.

## <optgroup>

Ignored. Options in a <select> element are rendered as a flat list on the screen.

## <option>

Specifies an item in a <select> list. <option> elements are rendered in the manner specified by the enclosing <select> list.

Attribute	Description
value	Specifies the value of the enclosing <select> variable, if the <select> element has a NAME attribute.
title	Ignored.
onpick	Specifies the URI that the browser loads when the option is selected.

## <select>

Presents a list to the user. Users can select one or more of the options.

Select lists are fully displayed on every card. Users can scroll through options. Single-selection lists are rendered as option buttons. Multiple-selection lists are rendered as check boxes.

See the <option> element for more information.

Attribute	Description
title	Ignored.
name	Specifies the name of the variable that is set with the selection.
value	Sets the default value of the variable.
iname	Specifies the variable to be set with the (1-based) index of the selected option, according to the specifications.
ivalue	Sets the index(es) of the preselected option(s). Use this attribute only if the variable specified by iname does not already have a value.
multiple	Specifies whether the <select> element is rendered as a multiple-selection list with a set of check boxes.
tabindex	Ignored. Tabs are not supported.

## Variable elements

### <setvar>

Specifies a new value for a given variable in the WAP context. When a <go>, <prev>, or <refresh> task is performed, the browser first looks for any associated <setvar> elements. It then updates the WAP context accordingly before running the task.

Attribute	Description
name	Sets the name of the variable.
value	Sets the value of the variable.

<timer>

Specifies a timer for the current card. The browser implements card timers according to the WML specifications. In particular, a `<refresh>` operation stops the timer, sets its corresponding variable to the current timer value, performs the refresh, and then resets and restarts the timer.

When the timer expires, the variable that the `name` attribute specifies is set to 0 before any `<ontimer>` tasks are run.

Attribute	Description
name	Specifies the variable name to be set with the timer value for card entry, exit, and timer-expire events.
value	Specifies the initial timer value for on-card entry events.



# JavaScript language reference

## Using JavaScript Supported JavaScript objects

### Using JavaScript

The BlackBerry Browser supports JavaScript 1.0, 1.1, 1.2, 1.3, and small subsets of JavaScript 1.4 and 1.5. The browser also supports the ECMA-262 ECMAScript Language Specification..

**i** **Note:** JavaScript is supported only on BlackBerry devices with at least 16 MB of memory.

The browser processes JavaScript that is run when the page is first rendered and JavaScript that is associated with control actions on the page. The JavaScript support manages any additional HTML and JavaScript content that the JavaScript produces and reads any auxiliary JavaScript support libraries that are referenced from the page.

**i** **Note:** The BlackBerry Browser does not support style sheets for Dynamic HTML. Any JavaScript on the page that creates Dynamic HTML effects (for example, pop-up menus) runs but has no visual effect, and might not be fully functional. The JavaScript support is provided for pages for which HTML content is partially or fully generated by JavaScript and for data processing operations that are coded in JavaScript, such as input field validation in forms and processing Login buttons on various sites.

On the BlackBerry Browser, users can turn JavaScript support on or off. JavaScript support can also be turned off through an IT policy.

See “Scripting Basics” on page 193 for information about JavaScript reserved words and supported statements, and operators and expressions.

Additional JavaScript resources can be found on the Internet.

### Supported JavaScript objects

The BlackBerry Browser supports the following JavaScript objects:

Object	Summary	See page
BlackBerry	The BlackBerry object defines the network read-only property, which is used to define the network on which the BlackBerry communicates.	134
BlackBerry Location	The BlackBerry Location object is designed to provide access to the GPS location of the BlackBerry device. The GPS location refers to the geographical co-ordinates, latitude and longitude, of the BlackBerry device.	135
Navigator	The Navigator object is designed to return information about the version of the BlackBerry Browser that is being used. All its properties, which are read-only, contain information about different aspects of the browser.	138

Object	Summary	See page
Document	The Document object provides access to the elements in an HTML page from within your script. This includes the properties of every form, link, and anchor (and, where applicable, any subelements), as well as global document properties such as background and foreground colors.	142
Form	The Form object lets you access the elements of an HTML form that is used to collect information from users.	149
Screen	The Screen object returns information about the dimensions and color depth of the BlackBerry device display.	156
Window	The Window object is created automatically when the browser encounters a <body> or <frame> tag, and returns information about the window.	158
Window History	The Window History object stores an array of the URLs previously visited by the user during the current browser session.	170

## BlackBerry

The BlackBerry object defines the network read-only property, which is used to define the network on which the BlackBerry communicates.

Member type	Member name	Notes	See page
Properties	location	Read-only.	135
	network	Read-only.	135

## location

Acts a pointer to the `blackberry.location` object. This property is supported only by BlackBerry devices running BlackBerry Device Software Version 4.1 or later. This property is read-only.

**Syntax** `blackberry.location`

**Parameters** none

**Returns** A pointer to the `blackberry.location` object.

**Example** The following code fragment displays the current geographic co-ordinates of the BlackBerry device:

```
document.write("The client BlackBerry device is currently
               located at " + blackberry.location.latitude +
               " degrees latitude and " +
               blackberry.location.longitude +
               " degrees longitude.");
```

**See also**

- `blackberry.location.GPSSupported`
- `blackberry.location.latitude`
- `blackberry.location.longitude`
- `blackberry.location.onLocationUpdate()`
- `blackberry.location.refreshLocation()`
- `blackberry.location.setAidMode()`

## network

Identifies the wireless network on which the BlackBerry device is communicating. This property is read-only.

**Syntax**      `blackberry.network`

**Returns**      A string identifying one of the following networks: Mobitex®, GPRS, CDMA, or iDEN.

**Example**      The following code fragment displays the value of the network property:

```
document.write("The client BlackBerry device is communicating
               on the " + blackberry.network + " wireless
               network.");
```

**See also**      `window.blackberry`

## BlackBerry Location

The BlackBerry Location object is designed to provide access to the GPS location of the BlackBerry device. The GPS location refers to the geographical co-ordinates, latitude and longitude, of the BlackBerry device.

Member type	Member name	Notes	See page
Methods	<code>onLocationUpdate()</code>	—	137
	<code>refreshLocation()</code>	—	137
	<code>setAidMode()</code>	—	137
Properties	<code>GPSSupported</code>	Read-only.	135
	<code>latitude</code>	Read-only.	139
	<code>longitude</code>	Read-only.	139

## GPSSupported

Returns whether the BlackBerry device supports GPS or not. This property is read-only.

**Syntax**      `blackberry.location.GPSSupported`

**Returns**

- Boolean `true` if the BlackBerry device supports GPS.
- Boolean `false` if the BlackBerry device does not support GPS.

**Example**      The following code determines whether GPS is supported, then sets the GPS location aid mode and refreshes the location:

```
if(blackberry.location.GPSSupported) {
    blackberry.location.setAidMode(0);
    blackberry.location.refreshLocation();
}
```

**See also**      `blackberry.location`

## latitude

Provides the current latitude of the BlackBerry device. To make sure that the most accurate co-ordinate is returned, you should call `refreshLocation()` first. This property is read-only.

**Syntax** `blackberry.location.latitude`

**Returns** The current latitude, in degrees, of the BlackBerry device. Positive values indicate northern latitude; negative values indicate southern latitude.

**Example** The following code fragment displays the current geographic co-ordinates of the BlackBerry device:

```
document.write("The client BlackBerry device is currently
               located at " + blackberry.location.latitude +
               " degrees latitude and " +
               blackberry.location.longitude +
               " degrees longitude.");
```

**See also** `blackberry.location`  
`blackberry.location.longitude`

## longitude

Provides the current longitude of the BlackBerry device. To make sure that the most accurate co-ordinate is returned, you should call `refreshLocation()` first. This property is read-only.

**Syntax** `blackberry.location.longitude`

**Returns** The current longitude, in degrees, of the BlackBerry device. Positive values indicate eastern longitude; negative values indicate western longitude.

**Example** The following code fragment displays the current geographic co-ordinates of the BlackBerry device:

```
document.write("The client BlackBerry device is currently
               located at " + blackberry.location.latitude +
               " degrees latitude and " +
               blackberry.location.longitude +
               " degrees longitude.");
```

**See also** `blackberry.location`  
`blackberry.location.latitude`



## onLocationUpdate()

Registers a callback method that is called when the location is updated using `refreshLocation()`.

**Syntax** `blackberry.location.onLocationUpdate(method)`

**Parameters** *method* A supported JavaScript method.

**Example** The following code fragment displays an alert to the user when the location is updated, informing them of the updated location.

```
blackberry.location.onLocationUpdate(window.alert("Your new
position is " + blackberry.location.latitude +
" degrees latitude and " +
blackberry.location.longitude +
" degrees longitude."));
```

**See also** `blackberry.location`

## refreshLocation()

Requests an update of the location of the BlackBerry device.

**Syntax** `blackberry.location.refreshLocation`

**Returns** A string containing the current latitude and longitude, in degrees, of the BlackBerry device.

**Example** The following code determines whether GPS is supported, then sets the GPS location aid mode and refreshes the location:

```
if(blackberry.location.GPSSupported) {
    blackberry.location.setAidMode(0);
    blackberry.location.refreshLocation();
}
```

**See also** `blackberry.location`

## setAidMode()

Specifies which method the BlackBerry device will use to obtain the GPS location. The device can obtain location information in one of three ways.

Aid Mode	Description
Cellsite	This method uses the GPS location of the active cellsite tower to provide first order GPS information. It provides the least accurate location information; however, it is the fastest location mode. <b>Note:</b> This location method requires network connectivity and carrier support.
Assisted	This method uses the network to provide ephemeris satellite data to the device chip. It provides the GPS location faster than the autonomous mode and more accurately than the cellsite mode. <b>Note:</b> This location method requires network connectivity and carrier support.
Autonomous	This method uses the GPS chip on the BlackBerry device without assistance from the network. The autonomous mode provides the first GPS location in the slowest amount of time.

<b>Syntax</b>	<code>blackberry.location.setAidMode(<i>aidMode</i>)</code>	
<b>Parameters</b>	<i>aidMode</i>	Identifies the method used to obtain the GPS location. The value for this parameter may be one of: <ul style="list-style-type: none"> <li>• 0 (Cellsite)</li> <li>• 1 (Assisted)</li> <li>• 2 (Autonomous)</li> </ul>
<b>Example</b>	The following code determines whether GPS is supported, then sets the GPS location method and refreshes the location: <pre>         if(blackberry.location.GPSSupported) {             blackberry.location.setAidMode(0);             blackberry.location.refreshLocation();         }     </pre>	
<b>See also</b>	<code>blackberry.location</code>	

## Navigator

The Navigator object is designed to return information about the version of the BlackBerry Browser that is being used. All its properties, which are read-only, contain information about different aspects of the browser.

Member type	Member name	Notes	See page
Methods	<code>javaEnabled()</code>	—	139
	<code>plugins.refresh()</code>	Stub implementation. This method has no effect.	—
	<code>preference()</code>	—	141
	<code>savePreferences()</code>	Stub implementation. This method has no effect.	—
	<code>taintEnabled()</code>	—	141
	<code>updateAppList()</code>	—	141
Properties	<code>appCodeName</code>	Read-only.	135
	<code>appName</code>	Read-only.	139
	<code>appVersion</code>	Read-only.	139
	<code>language</code>	Read-only.	140
	<code>mimeType</code>	Read-only.	140
	<code>platform</code>	Treated as a constant. Returns "BlackBerry."	—
	<code>plugins</code>	Not supported. Returns an empty array.	—
	<code>userAgent</code>	Read-only.	142

## appCodeName

Provides the application code name. This property is read-only.

**Syntax** `navigator.appCodeName`

**Returns** A string specifying the code name of the browser. The value could depend on the emulation mode.

**Example** The following code fragment displays the value of the appCodeName property:

```
document.write("The code name of this application is " +
    navigator.appCodeName);
```

## appName

Provides the name of the client browser. This property is read-only.

**Syntax** `navigator.appName`

**Returns** A string that specifies the name of the browser. In the case of the BlackBerry device, the value is always BlackBerry.

**Example** The following code fragment displays the value of the appName property, which, in the case of the BlackBerry device, will be BlackBerry:

```
document.write("The name of the client browser is " +
    navigator.appName);
```

## appVersion

Provides the version of the BlackBerry Browser. This property is read-only.

**Syntax** `navigator.appVersion`

**Returns** A string that contains the version of the BlackBerry Browser (for example, 4.1.0).

**Example** The following code fragment displays version information for the browser:

```
document.write("The version of the BlackBerry Browser is " +
    navigator.appVersion);
```

## javaEnabled()

Tests whether the browser supports Java or not.

**Syntax** `navigator.javaEnabled()`

**Returns** In the case of the BlackBerry Browser, always returns true.

**Example** The following code fragment runs the function doThis if Java is supported otherwise, it runs the function doThat:

```
if (navigator.javaEnabled()) {
    doThis();
}
else doThat();
```

## language

Provides the two-letter language code that represents the default language translation of the browser. This property is read-only.

**Syntax** `navigator.language`

**Returns** A string that contains the two-letter language code (For example, en).

**Example** The following code fragment displays the value of the language property:

```
document.write("The default language translation of the client
               browser is " + navigator.language);
```

## mimeTypes

Provides the MIME types that the client supports. This property is read-only.

**Syntax** `navigator.mimeTypes[ index ]`  
`navigator.mimeTypes.length`

**Parameters** *index* An integer that represents a MIME type contained in the array.

**Properties** *length* Specifies the number of MIME types that the client supports.

**Returns** An array of supported MIME types.

**Example** The following code fragment displays the type, description, and suffixes properties for each supported MIME type on a client:

```
for (i=0; i < navigator.mimeTypes.length; i++) {
    document.writeln("<TR VALIGN=TOP><TD>",i,"</TD>",
        "<TD>",navigator.mimeTypes[i].type,"</TD>",
        "<TD>",navigator.mimeTypes[i].desc,"</TD>",
        "<TD>",navigator.mimeTypes[i].suffixes,"</TD>");
}
```

## preference()

Queries the preferences of the BlackBerry device.



Note: The BlackBerry Browser does not support the `setValue` parameter.

**Syntax** `navigator.preference(prefName)`

**Parameters** *prefName* The name of the preference whose value you want to retrieve. Valid names include:

- `general.always_load_images`
- `security.enable_java`
- `javascript.enabled`
- `browser.enable_style_sheets`
- `autoupdate.enabled`
- `network.cookie.cookieBehavior`
- `network.cookie.warnAboutCookies`

**Returns** The value of the specified preference.

**Example** The following code fragment creates an image link if images are always loaded; otherwise it creates text link.

```
if (navigator.preference(general.always_load_images)) {
    document.writeln("<a href=\"www.blackberry.com\">
    <img src=\"bb_logo.gif\"></a>");
}
else {
    document.writeln("<a href=\"www.blackberry.com\">
    www.blackberry.com</a>");
}
```

## taintEnabled()

Tests whether data tainting is supported or not.

**Syntax** `navigator.taintEnabled()`

**Returns** In the case of the BlackBerry Browser, this method always returns Boolean false.

**Example** The following code fragment runs the function `doThis` if data tainting is supported; otherwise, it runs the function `doThat`.

```
if (navigator.taintEnabled()) {
    doThis();
}
else doThat();
```

## userAgent

Extracts the user agent from the user-agent header of the HTTP header. The user agent is used by servers to identify the client browser. This property is read-only.

**Syntax**      `navigator.userAgent`

**Returns**      A string that represents the current user-agent.

**Example**      The following code fragment displays user-agent information for the client browser:

```
document.write("The value of navigator.userAgent is " +
    navigator.userAgent);
```

For example, in the case of a BlackBerry Browser client, this property might return BlackBerry8700/4.1.0 (device model 8700, running BlackBerry Device Software Version 4.1.0).

## Document

The Document object provides access to the elements in an HTML page from within your script. This includes the properties of every form, link, and anchor (and, where applicable, any subelements), as well as global document properties, such as background and foreground colors.

Member type	Member name	Notes	See page
Methods	<code>captureEvents()</code>	Not supported.	—
	<code>close()</code>	—	144
	<code>contextual()</code>	Not supported.	—
	<code>getSelection()</code>	Not supported.	—
	<code>handleEvent()</code>	Not supported.	—
	<code>open()</code>	—	147
	<code>releaseEvents()</code>	Not supported.	—
	<code>routeEvents()</code>	Not supported.	—
	<code>write()</code>	—	148
	<code>writeln()</code>	—	149
Properties	<code>alinkColor</code>	Read-only.	143
	<code>anchors</code>	Not supported. Returns an empty array.	—
	<code>applets</code>	Not supported. Returns an empty array.	—
	<code>bgColor</code>	Read-only.	143
	<code>classes</code>	Not supported. Returns an empty array.	—
	<code>cookie</code>	Read-write.	144
	<code>domain</code>	Read-only.	145
	<code>embeds</code>	Not supported. Returns an empty array.	—
	<code>fgColor</code>	Read-only.	145
	<code>formName</code>	Read-only.	145
	<code>forms</code>	Read-only.	146

Member type	Member name	Notes	See page
Properties (continued)	height	Treated as a constant. Returns the height of the device screen, in pixels.	—
	ids	Not supported. Returns an empty array.	—
	images	Not supported. Returns an empty array.	—
	lastModified	Read-only.	146
	linkColor	Read-only.	146
	plugins	Not supported. Returns an empty array.	—
	referrer	Read-only.	147
	tags	Not supported. Returns an empty array.	—
	title	Read-write.	147
	URL	Read-only.	148
	vlinkColor	Read-only.	148
	width	Treated as a constant. Returns the width of the device screen, in pixels.	—

## alinkColor

Provides the color that is used to identify active links in the document. This property is read-only.

**Syntax** `document.alinkColor`

**Returns** A string that represents the active link color. Colors can be represented as a string literal (for example, teal) or as a hexadecimal triplet (for example, #0055FF).

**Example** The following code fragment returns the color that is used to identify active links in the active document:

```
document.write("Active links in this document are represented
using the color " + document.alinkColor);
```

**See also** `linkColor`  
`vlinkColor`

## bgColor

Returns the color that is used for the background of the active document. This property is read-only.

**Syntax** `document.bgColor`

**Returns** A string that represents the background color of the document. Colors can be represented as a string literal (for example, teal) or as a hexadecimal triplet (for example, #0055FF).

**Example** The following code fragment returns the color that is used as the background color in the active document:

```
document.write("The background color of this document is " +
document.bgColor);
```

**See also** `fgColor`

## close()

Closes a previously opened output stream.

**Syntax** `document.close()`

**Example** The following code fragment calls `document.close()` to close a stream that was opened with `document.open()`. The `document.close()` method forces the content of the stream to display in the window.

```

function doThis() {
    var string1 = "Hi mom!";
    var string2 = "Bye mom!";
    document.open();
    document.write(string1 + "<P>" + string2);
    document.close();
}

```

**See also** `open()`

## cookie

Provides a report that details all visible and unexpired cookies that are associated with the specified document. This is a writable property.

**Syntax** `document.cookie`

**Returns** A comma-separated list that contains the cookies for the active document.

**Examples** The following code fragment displays information about visible and unexpired cookies:

```

document.write("The following cookies are associated with this
               document:
document.writeln(document.cookie)

```

The following code fragment creates a cookie

```

function createCookie(name,value,days)
{
    if (days)
    {
        var date = new Date();
        date.setTime(date.getTime()+(days*24*60*60*1000));
        var expires = "; expires="+date.toGMTString();
    }
    else var expires = "";
    document.cookie = name+"="+value+expires+"; path=/";
}

```



## domain

Provides the domain of the server that served the active document. This property is read-only.

**Syntax** `document.domain`

**Returns** A string that contains the domain name of the server that served the document.

**Example** The following code fragment returns the domain name of the hosting server:

```
document.write("This document originated from the " +
    document.domain + "domain.");
```

## fgColor

Returns the color that is used for the foreground (the text) of the active document. This property is read-only.

**Syntax** `document.fgColor`

**Returns** A string that represents the foreground text color. Colors can be represented as a string literal (for example, teal) or as a hexadecimal triplet (for example, #0055FF).

**Example** The following code fragment returns the text color that is used in the active document:

```
document.write("The text color of this document is " +
    document.fgColor);
```

**See also** `bgColor`

## formName

Provides a specific form in the active document, referring to it by the Name attribute of the HTML <form> tag. This property is read-only.

**Syntax** `document.formName`

**Parameters** *formName* Represents the value of the Name attribute for the HTML <form> tag.

**Returns** The form associated with the given *formName*.

**Example** The following code fragment submits a form called loginForm:

```
document.loginForm.submit();
```

**See also** `forms`  
`Form`

## forms

Provides a list of all the form objects in the active document. This property is read-only.

**Syntax**      `document.forms[index]`  
                  `document.forms.length`

**Parameters**    *index*      An integer or string that represents a form that is contained in the array.

**Properties**      *length*      Specifies the number of forms contained in the active document.

**Returns**        An array that contains references to all the forms found in the active document.

**Example**        The following code fragment submits the first form in the forms array:

```
document.forms[0].submit();
```

**See also**        *formName*  
                  *Form*

## lastModified

Extracts the date that the document was last modified from the HTTP header. This property is read-only.

**Syntax**        `document.lastModified`

**Returns**        A string representing the date that the document was last modified.

**Example**        In the following code fragment, `lastModified` is used in a `<script>` tag at the end of an HTML file to display the modification date of the page:

```
document.write("This page updated on " +  
                 document.lastModified);
```

## linkColor

Returns the color that the active document uses to identify links. This property is read-only.

**Syntax**        `document.linkColor`

**Returns**        A string that represents the link color. Colors can be represented as a string literal (for example, `teal`) or as a hexadecimal triplet (for example, `#0055FF`).

**Example**        The following code fragment returns the color that is used as the link color in the active document:

```
document.write("Hyperlinks in this document are represented  
                 using the color " + document.linkColor);
```

**See also**        *alinkColor*  
                  *vlinkColor*

## open()

Opens the output stream to the current document to collect data from the write() and writeln() methods.

**Syntax** `document.open()`

**Example** The following code fragment calls document.close() to close a stream that was opened with document.open(). The document.close() method forces the content of the stream to display in the window.

```
function doThis() {
    var string1 = "Hi mom!";
    var string2 = "Bye mom!";
    document.open();
    document.write(string1 + "<P>" + string2);
    document.close();
}
```

**See also** `close()`

## referrer

Provides the URL of the document through which the user arrived at the active document, if it exists. This property is read-only.

**Syntax** `document.referrer`

**Returns**

- If the user arrived at the active document from another document, a string that contains the URL of the referring document is returned.
- If the user typed the URL or reached the active document through some other means, a null string is returned.

**Example** In the following code fragment, the getReferrer() function is called from the active destination document. It returns the URL of the source document:

```
function getReferrer() {
    return document.referrer;
}
```

## title

Retrieves or specifies the title of the active document. This is a writable property.

**Syntax** `document.title["title"]`

**Parameters** *title* The name of the document, as it appears in the title bar of the BlackBerry Browser.

**Returns** A string that contains the title of the document, as specified by the <title> element.

**Example** The following code fragment sets the title of the active document as "BlackBerry Home Page":

```
document.title="BlackBerry Home Page";
```

## URL

Retrieves the complete URL of the active document. This property is read-only.

**Syntax** `document.URL`

**Returns** A string that contains the URL of the active document.

**Example** The following code fragment displays the URL of the current document:

```
document.write("The current URL is " + document.URL);
```

**See also** `window.location`

## vlinkColor

Returns the color that the active document uses to identify links previously visited by the user. This property is read-only.

**Syntax** `document.vlinkColor`

**Returns** A string that represents the visited link color. Colors can be represented as a string literal (for example, teal) or as a hexadecimal triplet (for example, #0055FF).

**Example** The following code fragment returns the color that is used as the link color in the active document:

```
document.write("Visited links in this document are represented  
using the color " + document.vlinkColor);
```

**See also** `alinkColor`  
`linkColor`

## write()

Writes HTML expressions to the specified document.

**Syntax** `document.write(Expression1 [, Expression2, ...]);`

**Parameters** *Expression1* An HTML expression. Additional expressions can be included.

**Example** In the following code fragment, `write()` takes several arguments, including strings, a numeric, and a variable, and displays the string "Hello world testing 123":

```
var mystery = "world";  
msgWindow.document.write("Hello ", mystery, " testing ", 123);
```

In the following code fragment, `write()` takes two arguments to display the string "Hello world...." The first argument is an assignment expression, and the second argument is a string literal.

```
msgWindow.document.write(mystr = "Hello " + "world...");
```

In the following code fragment, the `write` method takes a single argument that is a conditional expression. If the value of the variable `age` is less than 18, the method displays "Minor." If the value of `age` is greater than or equal to 18, the method displays "Adult."

```
msgWindow.document.write(status = (age >= 18) ? "Adult" :  
"Minor");
```

**See also** `writeln()`

## writeln()

Writes HTML expressions to the specified document, and places a new line character at the end of the expression.

**Syntax**      `document.writeln(Expression1 [, Expression2, ...]);`

**Parameters**    *Expression1*    An HTML expression. Additional expressions can be included.

**Example**        The following code displays the type, description, and suffixes properties for each supported MIME type on a client:

```
for (i=0; i < navigator.mimeTypes.length; i++) {
  document.writeln("<TR VALIGN=TOP><TD>",i,
    "<TD>",navigator.mimeTypes[i].type,
    "<TD>",navigator.mimeTypes[i].description,
    "<TD>",navigator.mimeTypes[i].suffixes);
}
```

**See also**        `write()`

## Form

The Form object lets you access the elements of an HTML form that is used to collect information from users.

Member type	Member name	Notes	See page
Methods	<code>handleEvent()</code>	This method is not supported.	—
	<code>reset()</code>	—	154
	<code>submit()</code>	—	155
Event handlers	<code>onClick</code>	—	153
	<code>onReset</code>	—	153
	<code>onSubmit</code>	—	154
Properties	<code>action</code>	Read-write.	150
	<code>elements</code>	Read-only.	150
	<code>length</code>	Read-only.	151
	<code>method</code>	Read-only.	152
	<code>name</code>	Read-only.	152
	<code>target</code>	Read-write.	155

## action

Accesses the Action attribute of an HTML <form> element, which defines the URL to which the form is submitted. This is a writable property.

**Syntax** *formName*.action[="serverURL"]

**Parameters**

<i>formName</i>	The form for which data is being submitted. This parameter can be: <ul style="list-style-type: none"> <li>the value of the Name attribute for the HTML &lt;form&gt; tag</li> <li>an element in the document.forms array</li> </ul>
<i>serverURL</i>	The URL of the server to which form field input data is sent. This parameter can specify a Cell Global Identity (CGI) or LiveWire application on the server; it can also be a mailto: URL if the form is to be mailed.

**Returns** A string that describes the URL the form submits data to, unless serverURL is set.

**Example** The following code fragment specifies the URL to which loginForm submits data:

```
document.loginForm.action="urlName";
```

**See also** `document.formName`  
`document.forms`

## elements

Provides a list of the elements that are found in the form. This property is read-only.

**Syntax** *formName*.elements[ *index* ]  
*formName*.elements.length

**Parameters**

<i>formName</i>	The form that contains the listed elements. This parameter can be: <ul style="list-style-type: none"> <li>the value of the Name attribute for the HTML &lt;form&gt; tag</li> <li>an element in the document.forms array</li> </ul>
<i>index</i>	An integer that represents an object of the form, or the name of the element object as specified by its Name attribute.

**Properties** `length` See "length" on page 151 for more information.

**Returns** An array that contains each element of the form.

**Example** The following code fragment writes a list of the elements contained in the form loginForm:

```
document.write("This form contains the following HTML  
elements:");  
for (i=0; i < loginForm.elements.length; i++) {  
    document.write(loginForm.elements[i]);  
}
```

**See also** `length`  
`document.formName`  
`document.forms`

## encoding

Specifies the value of the Enctype attribute for the HTML <form> tag, which provides the MIME encoding of the form. This property is read-only.

**Syntax** *formName*.encoding

**Parameters** *formName* The form for which the encoding type is being retrieved. This parameter can be either

- the value of the Name attribute for the HTML <form> tag
- an element in the document.forms array

**Returns** A string that indicates the MIME encoding of the form.

**Example** The following function returns the value of the loginForm encoding property:

```
function getEncoding() {
    return document.loginForm.encoding;
}
```

**See also** `document.formName`  
`document.forms`

## length

Specifies the number of elements in the form. This property is read-only.

**Syntax**

- formName*.length
- formName*.elements.length

**Parameters** *formName* The form for which the number of elements is being retrieved. This parameter can be either

- the value of the Name attribute for the HTML <form> tag
- an element in the document.forms array

**Returns** An integer that represents the number of elements of the form.

**Example** The following code fragment writes a list of the elements contained in the form loginForm:

```
document.write("This form contains the following HTML
elements:");
for (i=0; i < loginForm.length; i++) {
    document.write(loginForm.elements[i]);
}
```

**See also** `elements`  
`document.formName`  
`document.forms`

## method

Returns the value of the Method attribute of the HTML <form> element, which defines the method used to submit the form data to the server. This property is read-only.

**Syntax** *formName*.method

**Parameters** *formName* The form for which the submit method is being queried. This parameter can be either

- the value of the Name attribute for the HTML <form> tag
- an element in the document.forms array

**Returns** A string that represents the submit method. This string can have a value of either GET or POST.

**Example** The following code fragment returns the submit method for the loginForm form:

```
document.write("This form submits data to the server using the  
" + loginForm.method + " method");
```

**See also** *document.formName*  
*document.forms*

## name

Returns the value of the Name attribute of the HTML <form> element, which provides the name of specified form. This property is read-only.

**Syntax** *formName*.name

**Parameters** *formName* The form for which the value of the Name attribute is being retrieved. This parameter is an element in the document.forms array.

**Returns** An array that contains each element of the form.

**Example** The following code fragment writes a list of each of the forms found in the active document:

```
document.write("This document contains the following HTML  
forms:");  
for (i=0; i < document.forms.length; i++) {  
    document.write(document.forms[i].name);  
}
```

**See also** *document.formName*  
*document.forms*



## onClick

Specifies the event that follows when an object on click event occurs.

**Syntax** *formName*.onClick[="event"]

**Parameters**

<i>formName</i>	The name of the form. This parameter can be either
	<ul style="list-style-type: none"> <li>the value of the Name attribute for the HTML &lt;form&gt; tag</li> <li>an element in the document.forms array</li> </ul>
<i>event</i>	The event that is performed.

**Returns** A string describing the event which has been previously specified to occur when a form object is clicked, unless *event* is specified.

**Example** The following code fragment shows an onclick event handler that calls the function isValidAddress() when the Calculate button is clicked.

```
<INPUT TYPE="button" VALUE="Calculate"
onClick="compute(this.form)">
```

**See also** document.*formName*  
document.forms

## onReset

Specifies the event that follows a reset action.

**Syntax** *formName*.onReset[="event"]

**Parameters**

<i>formName</i>	The name of the form. This parameter can be either
	<ul style="list-style-type: none"> <li>the value of the Name attribute for the HTML &lt;form&gt; tag</li> <li>an element in the document.forms array</li> </ul>
<i>event</i>	The event that is performed.

**Returns** A string describing the event which has been previously specified to occur when the form is reset, unless *event* is specified.

**Example** The following code fragment shows an onReset event handler that displays an alert dialog informing the user that default form values have been reset.

```
<FORM NAME="form1" onSubmit="return isValidAddress()"
onReset="alert('Default values have been
restored')">
  <B>Please enter your email address:</B>
  <INPUT TYPE="text" NAME="address" SIZE=10><P>
  <INPUT TYPE="submit" VALUE="Done" NAME="submit1">
  <INPUT TYPE="reset" VALUE="Clear Form" NAME="Reset1">
</FORM>
```

**See also** document.*formName*  
document.forms

## onSubmit

Specifies the event that follows a submit action.

**Syntax** *formName*.onSubmit[="event"]

**Parameters** *formName* The name of the form. This parameter can be either

- the value of the Name attribute for the HTML <form> tag
- an element in the document.forms array

*event* The event that is performed.

**Returns** A string describing the event which has been previously set to occur when the form is submitted, unless *event* is specified.

**Example** The following code fragment shows an onSubmit event handler that calls the function isValidAddress(), which determines whether the email address entered is valid before the form data is submitted to the server.

```
<FORM NAME="form1" onSubmit="return isValidAddress()"
      onReset="alert('Default values have been restored')">
  <B>Please enter your email address:</B>
  <INPUT TYPE="text" NAME="address" SIZE=10><P>
  <INPUT TYPE="submit" VALUE="Done" NAME="Submit">
  <INPUT TYPE="reset" VALUE="Clear Form" NAME="Reset">
</FORM>
```

**See also** document.*formName*  
document.forms

## reset()

Clears user input and resets the default values of the form.

**Syntax** *formName*.reset()

**Parameters** *formName* The name of the form to be reset. This parameter can be either

- the value of the Name attribute for the HTML <form> tag
- an element in the document.forms array

**Example** The following code fragment extracts data from the form located below. If the data is not entered properly (for example, CA or AZ is not entered), the form is reset.

```
function isValidAddress(){
    if (document.loginForm.address == .value == 'CA' ||
        textObject.value == 'AZ') {
        alert('Nice input');
    }
    else { document.loginForm.reset() }
```

**See also** document.*formName*  
document.forms

## submit()

Submits the specified form. This is equivalent to the user clicking a Submit button.

**Syntax** *formName*.submit()

**Parameters** *formName* The name of the form to be submitted. This parameter can be either

- the value of the Name attribute for the HTML <form> tag
- an element in the document.forms array

**Example** The following code fragment submits a form called loginForm:

```
document.loginForm.submit();
```

If loginForm is the first form you create, you also can submit it as follows:

```
document.forms[0].submit();
```

**See also** *document.formName*  
*document.forms*

## target

Specifies or returns the target window that responses are sent to after a form is submitted.



**Note:** Because the BlackBerry Browser is a single document interface (that is, it never displays more than one browser window at a time), setting a target essentially has no effect on content displayed in the BlackBerry Browser.

This is a writable property.

**Syntax** *formName*.target[="windowName"]

**Parameters** *formName* The name of the form to be submitted. This parameter can be either

- the value of the Name attribute for the HTML <form> tag
- an element in the document.forms array

*windowName* The name of the window to which responses are sent.

**Returns** A string that represents the name of the target window.

**Example** The following code fragment specifies that responses to the loginForm form are displayed in the msgWindow window:

```
document.loginForm.target="msgWindow";
```

## Screen

The Screen object returns information about the dimensions and color depth of the BlackBerry device display.

Member type	Member name	Notes	See page
Properties	availHeight	Read-only.	150
	availLeft	Treated as a constant. Returns 0.	—
	availTop	Treated as a constant. Returns 0.	—
	availWidth	Read-only.	150
	colourDepth	Read-only.	151
	height	Read-only.	152
	pixelDepth	Read-only.	152
	width	Read-only.	155

### availHeight

Retrieves the height of the BlackBerry device screen. This property behaves identically to the height property. This property is read-only.

**Syntax**      `screen.availHeight`

**Returns**      An integer that represents the screen height, in pixels.

**Example**      The following code fragment returns the screen height:

```
document.write("The screen is " + screen.availHeight + " pixels
high.");
```

**See also**      `availWidth`  
`height`

### availWidth

Retrieves the width of the BlackBerry device screen. This property behaves identically to the width property. This property is read-only.

**Syntax**      `screen.availWidth`

**Returns**      An integer that represents the screen width, in pixels.

**Example**      The following code fragment returns the screen width:

```
document.write("The screen is " + screen.availWidth + " pixels
wide.");
```

**See also**      `availHeight`  
`width`

## colourDepth

Retrieves the bit depth of the color palette. This property behaves identically to the pixelDepth property. This property is read-only.

**Syntax** `screen.colourDepth`

**Returns**

- The bit depth of the device color palette, if it uses one.
- If no palette is used, this property reflects the value of pixelDepth.

**Example** The following code fragment returns the bit depth of the color palette:

```
document.write("The BlackBerry device display has a pixel depth  
of " + screen.colourDepth);
```

**See also** `pixelDepth`

## height

Retrieves the height of the screen. This property behaves identically to the availHeight property. This property is read-only.

**Syntax** `screen.height`

**Returns** An integer that represents the screen height, in pixels.

**Example** The following code fragment returns the screen height:

```
document.write("The screen is " + screen.height + " pixels  
high.");
```

**See also** `availHeight`

## pixelDepth

Returns the bit depth of the palette. This property behaves identically to the colourDepth property. This property is read-only.

**Syntax** `screen.pixelDepth`

**Returns** An integer that represents the color resolution, in bits per pixel, of the display.

**Example** The following code fragment returns the screen height:

```
document.write("The screen has a pixel depth of " +  
screen.pixelDepth + " bits per pixel.");
```

**See also** `colourDepth`

## width

Retrieves the width of the screen. This property behaves identically to the `availWidth` property. This property is read-only.

**Syntax** `screen.width`

**Returns** An integer that represents the width of the screen, in pixels.

**Example** The following code fragment returns the screen width:

```
document.write("The screen is " + screen.width + " pixels
wide.");
```

**See also** `availWidth`

## Window

The Window object is created automatically when the browser encounters a `<body>` or `<frame>` tag, and returns information about the window.

Member type	Member name	Notes	See page
Methods	<code>alert()</code>	—	160
	<code>atob()</code>	—	160
	<code>back()</code>	—	160
	<code>blur()</code>	Stub implementation. This method has no effect.	—
	<code>btoa()</code>	—	161
	<code>captureEvents()</code>	Not supported.	—
	<code>clearInterval()</code>	—	161
	<code>clearTimeout()</code>	—	162
	<code>close()</code>	—	162
	<code>confirm()</code>	—	163
	<code>disableExternalCapture()</code>	Not supported.	—
	<code>enableExternalCapture()</code>	Not supported.	—
	<code>find()</code>	Not supported.	—
	<code>focus()</code>	Not supported.	—
	<code>back()</code>	—	170
	<code>handleEvent()</code>	Not supported.	—
	<code>home()</code>	—	165
	<code>moveBy()</code>	Stub implementation. This method has no effect.	—
	<code>moveTo()</code>	Stub implementation. This method has no effect.	—
	<code>open()</code>	—	166
	<code>print()</code>	Stub implementation. This method has no effect.	—
	<code>prompt()</code>	—	167
	<code>releaseEvents()</code>	Not supported.	—
	<code>resizeBy()</code>	Stub implementation. This method has no effect.	—

Member type	Member name	Notes	See page
Methods (continued)	resizeTo()	Stub implementation. This method has no effect.	—
	routeEvent()	Not supported.	—
	scroll()	Stub implementation. This method has no effect.	—
	scrollBy()	Stub implementation. This method has no effect.	—
	scrollTo()	Stub implementation. This method has no effect.	—
	setHotKeys()	Stub implementation. This method has no effect.	—
	setInterval()	—	168
	setTimeout()	—	168
	setZOptions()	Stub implementation. This method has no effect.	—
	stop()	—	169
Properties	blackberry	Read-only.	161
	closed	Treated as a constant. Returns Boolean false.	—
	crypto	Not supported.	—
	defaultStatus	Read-write.	163
	document	Read-only.	163
	frames	Read-only.	164
	history	Read-write.	165
	innerHeight	Treated as a constant. Returns the screen height.	—
	innerWidth	Treated as a constant. Returns the screen width.	—
	length	Read-only.	166
	location	Read-only.	166
	locationbar	Not supported.	—
	menubar	Not supported.	—
	name	Not supported.	—
	offscreenBuffering	Treated as a constant. Returns Boolean false.	—
	opener	Not supported.	—
	outerHeight	Treated as a constant. Returns the screen height.	—
	outerWidth	Treated as a constant. Returns the screen width.	—
	pageXOffset	Treated as a constant. Returns 0.	—
	pageYOffset	Treated as a constant. Returns 0.	—
	parent	Read-write.	167
	personalbar	Not supported.	—
	screenX	Treated as a constant. Returns 0.	—
	screenY	Treated as a constant. Returns 0.	—
	scrollbars	Not supported.	—
	self	Read-write.	167
	stop()	Not supported.	—
	statusbar	Not supported.	—
	toolbar	Not supported.	—
	top	Read-write.	169
	window	Read-write.	169

## alert()

Displays a standard alert dialog box with an OK button.

**Syntax** `window.alert("message")`

**Parameters** *message* A string, or a property of an existing object, that is displayed as the dialog box message.

**Example** The following code fragment displays an alert dialog box informing users that they did not enter information properly.

```
        window.alert("Please enter a name that is 8 characters or  
                    less");
```

**See also** `confirm()`  
`prompt()`

## atob()

Decodes a given Base64 string.

**Syntax** `window.atob(string)`

**Parameters** *string* The string to be decoded.

**Example** The following code fragment decodes a variable named `encodedURL` into Base64 and assigns the result to the variable `decodedURL`:

```
        var decodedURL=window.atob(encodedURL);
```

**See also** `btoa()`

## back()

Displays the previous URL in the history list. This method is equivalent to the user clicking Back in the BlackBerry Browser menu or clicking the Escape key during a browser session.

**Syntax** `window.back()`

**Example** The following code fragment adds a custom button to an HTML page that displays the previous item in the history list:

```
<INPUT TYPE="button" VALUE="Back" onClick="window.back()">
```

**See also** `close()`  
`forward()`  
`history`  
`window.history.back()`  
`window.history.previous`



## blackberry

Acts a pointer to the BlackBerry object. This property is read-only.

**Syntax** `window.blackberry`

**Returns** A pointer to the BlackBerry object.

**Example** The following code fragment displays the value of the network property:

```
document.write("The client BlackBerry device is communicating  
on the " + window.blackberry.network + " wireless  
network.");
```

**See also** BlackBerry

## btoa()

Encodes a given string to Base64.

**Syntax** `window.btoa(string)`

**Parameters** *string* The string to be encoded.

**Example** The following code fragment encodes a URL into Base64 and assigns the result to the variable encodeURL:

```
var encodeURL=window.btoa(URL)
```

**See also** atob()

## clearInterval()

Clears a repeating timer.

**Syntax** `window.clearInterval(intervalID)`

**Parameters** *intervalID* The ID (specified when the interval timer was created using setInterval()) of the interval to be cleared.

**Example** After a user clicks the Start button, the following code fragment writes the given string every five seconds (5,000 milliseconds). The user must click the Stop button to cancel the timer and stop new lines from being written.

```
<input type="button" value="Start"  
      onclick="timerID=setInterval(document.writeln("the  
timer is on"), 5000);" />  
<input type="button" value="Stop"  
      onclick="timerID=clearInterval(timerID);" />
```

**See also** setInterval()

## clearTimeout()

Clears a nonrepeating timer.

**Syntax** `window.clearTimeout(timeoutID)`

**Parameters** *timeoutID* The identifier (set using `setTimeout()`) that specifies the timeout evaluation that is cleared.

**Example** The following code fragment runs the `displayAlert()` function five seconds (5,000 milliseconds) after the user clicks a button. If the user clicks the second button before the message is displayed, the timeout is canceled and the message is not displayed.

```

<INPUT TYPE="button" VALUE="5-second reminder"
      NAME="remind_button"
      onClick="timerID=setTimeout('displayAlert()',5000)"
>
<INPUT TYPE="button" VALUE="Clear the 5-second reminder"
      NAME="remind_disable_button"
      onClick="clearTimeout(timerID)">

```

**See also** `setTimeout()`

## close()

Closes the active window, going back one element in history.



**Note:** Because the BlackBerry Browser is a single document interface, there are no active or inactive windows; instead, it stacks windows as successive items in the history list. If a user closes the current window, the previous document appears. The only exception is when the current document is the first item in the history. In this case, closing the window causes users to exit the browser.

**Syntax** `window.close()`

**Example** The following code fragment closes the active window and displays the previous page in the history list:

```
window.close()
```

**See also** `open()`

## confirm()

Displays a standard confirmation dialog box with an OK button and a Cancel button.

**Syntax** `window.confirm("message")`

**Parameters** *message* A string, or a property of an existing object, that is displayed as the confirmation message.

**Returns**

- If users click OK, the method returns `true`.
- If users click Cancel, the method returns `false`.

**Example** The following code fragment displays a confirmation dialog box asking users to confirm that they want to submit a form:

```
        window.confirm("Are you sure you want to submit this  
                        information?")
```

**See also** `alert()`  
`prompt()`

## defaultStatus

Defines the default message that is displayed in the status bar. You can override this message using the `status` property. This is a writable property.

**Syntax** `window.defaultStatus = "message"`

**Parameters** *message* The message that is displayed by default in the status bar.

**Example** The following code fragment sets the default status message:

```
        window.defaultStatus = "Click the link to go to the BlackBerry  
                                homepage.";
```

**See also** `stop()`

## document

Specifies the Document object that is contained within the window. This property is read-only.

**Syntax** `window.document.propertyName`  
`window.document.methodName`

**Parameters** *propertyName* The `defaultStatus`, `status`, `length`, or `name` property of the Document object.

*methodName* Any method associated with the Document object.

**Example** In the following code fragment, the `write()` method takes several arguments, including strings, a numeric, and a variable, and displays the string "Hello world testing 123":

```
        var mystery = "world" msgWindow.document.write("Hello ",  
                                                        mystery, " testing ", 123);
```

**See also** `Document`

## forward()

Displays the next element in the history list of the browser. This method is equivalent to the user clicking Forward in the BlackBerry Browser menu.

**Syntax** `window.forward()`

**Example** The following code adds a custom button to an HTML page that displays the next item in the history list:

```
<INPUT TYPE="button" VALUE="Forward"
      onClick="window.forward()">
```

**See also** `back()`  
`history`  
`window.history.forward()`  
`window.history.next`

## frames

Specifies the frames that the current document contains. This property is read-only.



**Note:** The BlackBerry Browser renders framesets by stacking frames vertically on a single page in the order in which they are encountered, using the full width of the device screen and as much vertical space as is required to contain all the frames.

**Syntax** `window.frames[index]`

**Parameters** *index* An integer that represents a child frame in the frameset, or the name of the Frame object as specified by its name attribute.

**Returns** An array that contains the frame(s) associated with the document.

**Example** The following code fragment writes a list of the frames that are part of the current document:

```
document.write("You have visited the following URLs during this
               session:");
for (i=0; i < window.length; i++) {
    document.write(window.frames[i]);
}
```

## history

Retrieves an array of recently accessed URLs from the History object. This property is read-only.

**Syntax**      `window.history[ index ]`  
                  `window.history.propertyName`  
                  `window.history.methodName`

**Parameters**

<i>index</i>	An integer that represents a URL contained in the history list.
<i>propertyName</i>	The current, length, next, or previous property associated with the History object.
<i>methodName</i>	The back(), forward(), or go() method associated with the History object.

**Returns**      An array that contains each element in the history list.

**Example**      The following code fragment writes a list of the URLs that the user has visited during the current session:

```
document.write("You have visited the following URLs during this
               session:");
for (i=0; i < window.history.length; i++) {
    document.write(window.history[i]);
}
```

**See also**      back()  
                  forward()  
                  window.history.back()  
                  window.history.forward()  
                  window.history.go()

## home()

Returns to the page that the user set as the browser home page.

**Syntax**      `window.home()`

**Example**      The following code fragment displays the home page, as it is specified in the browser settings:

```
<INPUT TYPE="button" VALUE="Go to your homepage"
      onClick="window.self.home()">
```

**See also**      back()  
                  forward()

## location

Retrieves or specifies the URL of the current window. This is a writable property.

**Syntax** *windowReference.location[.propertyName]*  
*windowReference.location.methodName(parameters)*

**Parameters**

<i>windowReference</i>	The name of a window, or a synonym such as top or parent.
<i>propertyName</i>	The defaultStatus, status, length, or name property of the window object.
<i>methodName</i>	Any method associated with the window object.

**Returns** A string containing the URL that is currently displayed in the specified window.

**Example** The following code fragment points the top window to the given URL:

```
top.location="http://www.blackberry.com/";
```

**See also** `document.URL`

## length

Returns the number of frames in a parent window. This property is read-only.



**Note:** The BlackBerry Browser renders framesets by stacking frames vertically on a single page in the order in which they are encountered, using the full width of the device screen and as much vertical space as is required to contain all the frames.

**Syntax** *windowReference.length*

**Parameters** *windowReference* The name of a window, or a synonym such as top or parent.

**Returns** An array containing the frame(s) that are associated with the document.

**Example** The following code fragment writes a list of the frames that are part of the current document:

```
document.write("This page contains the following frames:");
for (i=0; i < window.length; i++) {
    document.write(window.frames[i]);
}
```

## open()

Opens a new browser window.

**Syntax** [*windowName*]=[*window.*]open(*URL*)

**Parameters**

<i>windowName</i>	The name of the new window.
<i>URL</i>	A string that represents the URL to be displayed in the child window.

**Example** The following code fragment opens a new window that displays the BlackBerry web site:

```
newWin=window.open("http://www.blackberry.com");
```

**See also** `close()`

## parent

A reference to the parent window. If no parent window exists, this property points to the current active window. This is a writable property.

**Syntax**      `parent.propertyName`  
                  `parent.methodName`

**Parameters**    *propertyName*    Any property that is associated with the Window object.  
                          *methodName*      Any method that is associated with the Window object.

**Example**        The following code fragment closes the parent window when users click a button:

```
<INPUT TYPE="button" VALUE="Close the parent window"
      onClick="window.parent.close()">
```

**See also**        `self`  
                  `window`

## prompt()

Displays a prompt dialog box that prompts users for input.

**Syntax**        `window.prompt(message [, inputDefault])`

**Parameters**    *message*            A string, or a property of an existing object, that is displayed as the prompt message.  
                          *inputDefault*    Any string that represents the default value of the input field.

**Example**        The following code fragment displays a dialog box that prompts users to specify the number of donuts they want, and sets the default number to 12.

```
mainWindow.prompt("Enter the number of donuts you want to
      order:", 12);
```

**See also**        `alert()`  
                  `confirm()`

## self

A pointer to the current active window. This is a writable property.

**Syntax**        `self.propertyName`  
                  `self.methodName`

**Parameters**    *propertyName*    Any property that is associated with the Window object.  
                          *methodName*      Any method that is associated with the Window object.

**Example**        The following code fragment closes the current active window:

```
self.close();
```

**See also**        `parent`  
                  `window`

## setInterval()

Sets a timer that lets you evaluate an expression at regular intervals.

**Syntax** `intervalID=window.setInterval(expression, msec)`

**Parameters** *intervalID* An identifier that is used only to cancel the timeout evaluation with `clearInterval()`.

*expression* The string expression or property of an existing object to be evaluated.

*msec* The time in milliseconds after which the expression is evaluated.

**Example** The following code fragment writes the given string every five seconds (5,000 milliseconds) after the user clicks the Start button. Users must click the Stop button to cancel the timer and stop new lines from being written.

```
<input type="button" value="Start"
      onclick="timerID=setInterval(document.writeln("the
      timer is on"), 5000);" />
<input type="button" value="Stop"
      onclick="timerID=clearInterval(timerID);" />
```

**See also** `clearInterval()`

## setTimeout()

Sets a non-repeating timer that lets you evaluate an expression after a specified amount of time.

**Syntax** `timeoutID=window.setTimeout(expression, msec)`

**Parameters** *timeoutID* An identifier that is used only to cancel the timeout evaluation with `clearTimeout()`.

*expression* The string expression or property of an existing object to be evaluated.

*msec* The time in milliseconds after which the expression is evaluated.

**Example** The following code fragment writes the given string five seconds (5,000 milliseconds) after the user clicks a button. If the user clicks the second button before the string is displayed, the timeout is canceled and the string is not written.

```
<INPUT TYPE="button" VALUE="Start"
      onClick="timerID=setTimeout(document.writeln("five
      seconds have passed"),5000)">
<INPUT TYPE="button" VALUE="Clear"
      onClick="clearTimeout(timerID)">
```

**See also** `clearTimeout()`



## stop()

Stops the current download. This method is equivalent to the user clicking Stop in the BlackBerry Browser menu.

**Syntax**      `window.stop()`

**Example**      The following code fragment adds a button that lets users stop the current download.

```
<input type=button value="STOP" onClick="stop();">
```

## top

Contains a reference to the top window. Access to `window.top` provides only a URL, not the Window object itself.

Because the BlackBerry Browser is a single document interface, it attempts to maintain some idea of window relationships. In many cases, `window.top` points to the current window and is therefore equivalent to the `window` property; however, if users arrived at the current window from a frameset, `window.top` points to the parent frameset.

This is a writable property.

**Syntax**      `top.propertyName`  
                `top.methodName`

**Parameters**    *propertyName*    Any property that is associated with the Window object.  
                  *methodName*    Any method that is associated with the Window object.

**Example**      The following code fragment sets the background color of a document called `myDocument` to red.

```
top.myDocument.backgroundColor="red";
```

**See also**      `self`  
                `window`

## window

Contains a reference to the current window. Using the `window` property lets you invoke methods or call properties on the current window without confusion when multiple browser windows are open.

Because the BlackBerry Browser is a single document interface, a new window is opened each time a new URL is opened.

This is a writable property.

**Syntax**      `window.propertyName`  
                `window.methodName`

**Parameters**    *propertyName*    Any property that is associated with the Window object.  
                  *methodName*    Any method that is associated with the Window object.

**See also**      `self`  
                `top`

## Window History

The Window History object stores an array of the URLs that the user visited during the current browser session.

Members	Name	Notes	See page
Methods	<code>back()</code>	—	170
	<code>forward()</code>	—	171
	<code>go()</code>	—	171
Properties	<code>current</code>	Read-only.	170
	<code>length</code>	Read-only.	171
	<code>next</code>	Read-only.	172
	<code>previous</code>	Read-only.	172

### `back()`

Displays the previous URL in the history list of the browser. This method is equivalent to the user clicking Back in the BlackBerry Browser menu or clicking the Escape key during a browser session.

**Syntax** `window.history.back()`

**Example** The following code fragment adds a custom button to an HTML page that displays the previous item in the history list:

```
<INPUT TYPE="button" VALUE="Back"
      onClick="window.history.back()">
```

**See also** `forward()`  
`previous`  
`window.back()`  
`window.close()`  
`window.history`

### `current`

Returns the complete URL of the current history entry. This property is read-only.

**Syntax** `window.history.current`

**Returns** The complete URL of the current history entry.

**Example** The following code fragment displays the URL of the current item in the history list:

```
document.write("You are currently visiting the following
              URL:");
document.write(window.history.current);
```

**See also** `window.history`

## forward()

Displays the next element in the history list of the browser. This method is equivalent to the user clicking Forward in the BlackBerry Browser menu.

**Syntax** `window.history.forward()`

**Example** The following code adds a custom button to an HTML page that displays the next item in the history list:

```
<INPUT TYPE="button" VALUE="Forward"
      onClick="window.history.forward()">
```

**See also** `back()`  
`next`  
`window.back()`  
`window.history`

## go()

Causes the browser to display the URL that is the specified number of URLs before or after the current item in the history list.

**Syntax** `window.history.go(number)`

**Parameters** *number* The position of the new URL in the history list relative to the current item. A positive value moves forward in the history list; a negative value moves backward.

**Example** The following code fragment adds a custom button that causes the browser to display the URL three positions earlier than the current URL in the history list:

```
<INPUT TYPE="button" VALUE="Leap back three pages!"
      onClick="window.history.go(-3)">
```

**See also** `window.history`

## length

Returns the number of URLs in the history list. This property is read-only.

**Syntax** `window.history.length`

**Returns** An array containing the frame(s) that are associated with the document.

**Example** The following code fragment writes the URLs currently in the history list:

```
document.write("You have visited the following URLs during this
               session:");
for (i=0; i < window.history.length; i++) {
    document.write(window.history[i]);
}
```

**See also** `window.history`

## next

Retrieves the URL of the next entry in the history list. This property is read-only.

**Syntax** `window.history.next`

**Returns** The next entry in the history list. If this property returns a null value, the current URL is the last item in the list.

**Example** The following code adds a custom button to an HTML page that displays the next item in the history list:

```

var nextURL = window.history.next;
if(nextURL <> ""){
    document.write("The Forward menu item will take you to" +
        nextURL);
}
else {
    document.write("The current URL is the last item in the
        history list.")
}

```

**See also** `forward()`  
`window.history`

## previous

Retrieves the URL of the previous entry in the history list. This property is read-only.

**Syntax** `window.history.previous`

**Returns** The next entry in the history list. If this property returns a null value, the current URL is the first item in the list.

**Example** The following code adds a custom button to an HTML page that displays the next item in the history list:

```

var previousURL = window.history.previous;
if(previousURL <> ""){
    document.write("The Back button will take you to" +
        PreviousURL);
}
else {
    document.write("The current URL is the first item in the
        history list.")
}

```

**See also** `back()`  
`window.history`

# WMLScript language reference

## Using WMLScript WMLScript libraries

### Using WMLScript

WML is a broad, fairly easy-to-use language that permits content developers to create reasonable content that can be viewed on all WAP browsers. Unfortunately, it lacks many aspects of a true scripting language.

The solution is WMLScript, a wireless scripting language that can co-exist with WML decks. It is similar to JavaScript, and is a modified subset of ECMAScript.

WMLScripts are independent files called as external references within WML decks. They are compiled into bytecode at runtime on the server before they are sent to the WAP browser. Like C, C++, and Java, the language is case sensitive and has a construct that is similar to C.

The format of a WMLScript function is as follows:

```
extern function NAME( [PARAMETERS(S)] )  
{  
    // body of function  
}
```

The `extern` keyword makes the function public to external files. Do not include `extern` on functions that are only called from within the script.

Parameter passing is fairly lax where type is not specified in the parameter list. The caller of the function is responsible for making sure that parameters passed to a function are in the expected format and sequence.



**Note:** The BlackBerry Browser currently does not support floating-point values.

See “Scripting Basics” on page 193 for information about WMLScript reserved words and supported statements, operators, and expressions.

## WMLScript libraries

The strength of WMLScript is largely contained with the function libraries. The libraries include functions to deal with numeric values, dialog boxes and alerts, strings, relative and absolute URLs, and the browser.

The following WMLScript libraries are supported:

Library	Description	See page
Lang	The Lang library contains 15 core library functions. For example, you can perform operations on integers, create random numbers, create absolute values, stop code that is currently running, and determine support parameters.	174
Dialogs	The Dialogs library contains three dialog box handlers that are used to display alert, confirmation, and prompt dialog boxes. Users must respond to the displayed message or prompt in order for the SCRIPT to continue running.	178
String	The String library contains 14 functions that can be used to manipulate strings.	179
URL	The URL library contains 14 functions that manipulate URLs. For example, you can extract the various portions of the URL, escape and unescape special characters in the URL, determine the referring URL, and so on.	185
Browser	The Browser library contains seven functions that can be used to access the information that is associated with the WML cards. For example, you can return to a previously viewed card, go to a new card, refresh a card, and so on.	190

### Lang

The Lang library contains 15 core library functions. For example, you can perform operations on integers, create random numbers, create absolute values, stop code that is currently running, and determine support parameters.

Function	Description	See page
<code>abort()</code>	Aborts the WMLScript and returns the passed string to the caller.	175
<code>abs()</code>	Returns the absolute value of the passed number.	175
<code>characterSet()</code>	Returns a value that identifies the supported character set.	175
<code>exit()</code>	Exits the script and returns the passed message to the caller.	175
<code>isInt()</code>	Tests whether the passed string can be converted into an integer value using <code>parseInt()</code> .	176
<code>max()</code>	Determines the maximum value of two passed values in either integer or floating-point form.	176
<code>maxInt()</code>	Returns the maximum value of an integer.	176
<code>min()</code>	Returns the minimum value of two passed values in either integer or floating-point form.	176
<code>minInt()</code>	Returns the minimum value of an integer.	177
<code>parseInt()</code>	Converts a string into an integer value.	177
<code>random()</code>	Returns a random number between 0 and the passed value.	177
<code>seed()</code>	Initializes the random number generator.	177

## abort()

Aborts the WMLScript and returns the passed string to the caller.

**Syntax** `Lang.abort(ErrorMessage);`

**Parameters** *ErrorMessage* An error message that indicates the reason the script was aborted.

**Returns** No return value.

**Example** The following code fragment aborts the script and displays the given string:

```
Lang.abort("Script failure on line 123");
```

## abs()

Returns the absolute value of the passed number.

**Syntax** `Lang.abs(Number);`

**Parameters** *Number* An integer or float value.

**Returns** The absolute value returned as an integer or float value.

**Example** The following code fragment returns positive "98765":

```
var i_ret = Lang.abs(-98765);
```

The following code fragment returns positive "987.65":

```
var f_ret = Lang.abs(-987.65);
```

## characterSet()

Returns a value that identifies the supported character set.

Visit <http://www.iana.org/assignments/character-sets> for a current list of character sets.

**Syntax** `Lang.characterSet();`

**Returns** The numeric character-set identifier.

**Example** The following code fragment returns "3," representing US-ASCII:

```
var charset = Lang.characterSet();
```

## exit()

Exits the script and returns the passed message to the caller.

**Syntax** `Lang.exit(Message);`

**Parameters** *Message* A message to pass back to the caller.

**Returns** No return value.

**Example** The following code fragment exits the script:

```
Lang.exit("Exit stage left");
```

## isInt()

Tests whether the passed string can be converted into an integer value using `parseInt()`.

**Syntax** `Lang.isInt(StringValue);`

**Parameters** *StringValue* A string representation of an integer value.

**Returns**

- Boolean true if *StringValue* converts to integer form.
- Boolean false if *StringValue* cannot be converted.

**Example** The following code fragments return "true":

```
isOkay1 = Lang.isInt("98765");
isOkay2 = Lang.isInt("-98765");
isOkay3 = Lang.isInt("9.8e2");
```

The following code fragment returns "false":

```
isOkay4 = Lang.isInt("intni");
```

## max()

Determines the maximum value of two passed values in either integer or floating-point form.

**Syntax** `Lang.max(Value1, Value2);`

**Parameters** *Value1* The first of two integers or floating-point values to be compared.

*Value2* The second of two integers or floating-point values to be compared.

**Returns** The highest value passed.

**Example** The following sample returns "13":

```
var maxValue = Lang.max(12, 13);
```

## maxInt()

Returns the maximum value of an integer.

**Syntax** `Lang.maxInt();`

**Returns** The maximum integer value.

**Example** The following code fragment returns "2147483647":

```
var theMax = Lang.maxInt();
```

## min()

Returns the minimum value of two passed values in either integer or floating-point form.

**Syntax** `Lang.min(Value1, Value2);`

**Parameters** *Value1* The first of two integer or floating-point values to be compared.

*Value2* The second of two integer or floating-point values to be compared.

**Returns** The smallest value passed.

**Example** The following code fragment returns "12":

```
var theMin = Lang.min(12, 13);
```



**minInt()**

Returns the minimum value of an integer.

**Syntax**      `Lang.minInt();`

**Returns**      The minimum integer value.

**Example**      The following code fragment returns "-2147483648":

```
var theMin = Lang.minInt();
```

**parseInt()**

Converts a string into an integer value.

**Syntax**      `Lang.parseInt(StringInt);`

**Parameters**   *StringInt*    Integer value in string form.

**Returns**      Integer value.

**Example**      The following code fragment returns "9876":

```
var theInt1 = Lang.parseInt("9876");
```

The following code fragment stops parsing on the first error that it encounters, and therefore returns "987":

```
var theInt2 = Lang.parseInt("9876Hi!!");
```

**random()**

Returns a random number between 0 and the passed value.

**Syntax**      `Lang.random(MaxRange);`

**Parameters**   *MaxRange*    Maximum integer value to draw from.

**Returns**      A random integer within the specified range.

**Example**      The following code fragment returns an arbitrary value between 0 and 9867:

```
var rndValue = Land.random(9867);
```

**seed()**

Initializes the random number generator.

**Syntax**      `Lang.seed(SeedValue);`

**Parameters**   *SeedValue*    An integer seed value.

**Returns**      A null string.

**Example**      The following code fragment initializes the random number generator with an initial value of 98765:

```
var ret = Lang.seed(98765);
```

## Dialogs

The Dialogs library contains three dialog box handlers that are used to display alert, confirmation, and prompt dialog boxes. Users must respond to the displayed message or prompt for the script to continue running.

Function	Description	See
<code>alert()</code>	Displays a standard alert dialog box with an OK button.	178
<code>confirm()</code>	Displays a standard confirmation dialog box with an OK button and a Cancel button.	178
<code>prompt()</code>	Displays a prompt dialog box that prompts users for input.	179

### alert()

Displays a standard alert dialog box with an OK button.

**Syntax** `Dialogs.alert(Message);`

**Parameters** *Message* A string containing the message to display.

**Returns** A null string.

**Example** The following code fragment displays a dialog box with the message "Wake up now!":

```
var ret = Dialogs.alert("Wake up now!");
```

### confirm()

Displays a standard confirmation dialog box with an OK button and a Cancel button.

**Syntax** `Dialogs.confirm(Message, Button1, Button2);`

**Parameters** All parameters are strings or string literals.

*Message* The confirmation message.

*Button1* The text of the first dialog box button.

*Button2* The text of the second dialog box button.

**Returns**

- Boolean true if the user selects *Button1*.
- Boolean false if the user selects *Button2*.

**Example** The following example displays a dialog box with two buttons, Yes and No, and contains the message "Exit?":

```
var isOkay = Dialogs.confirm ("Exit?", "Yes", "No");
```

## prompt()

Displays a prompt dialog box that prompts users for input.

**Syntax** `Dialogs.prompt(Message, Default);`

**Parameters** All parameters are string or string literals.

*Message* The confirmation message.

*Default* The default response.

**Returns** A string response.

**Example** The following code fragment displays a dialog box that asks users their age. The default age is 30:

```
var Age = Dialogs.prompt("Your age", "30");
```

## String

The String library contains 14 functions that can be used to manipulate strings.

The length of a string is determined by its total number of characters, including white space. Each character in the string has an index position, where the first character is at position zero. A string can also be composed of a series of elements that are delineated by separators. A separator can be any character.

The six types of white space are blank space, carriage return, form feed, line feed, horizontal tab, and vertical tab.

Function	Description	See page
<code>charAt()</code>	Returns a single character that is located at the passed offset position within the passed string.	180
<code>compare()</code>	Compares strings where ranking is performed based on the ASCII value of each character within the string.	180
<code>elementAt()</code>	Locates a single element within the passed string buffer.	181
<code>elements()</code>	Counts how many times a delimiter occurs within a passed buffer to determine the number of elements in the buffer.	181
<code>find()</code>	Searches for the first occurrence of the passed substring within the passed buffer.	181
<code>format()</code>	Formats the passed numeric value as a string.	182
<code>insertAt()</code>	Creates a new string from the passed buffer that includes the passed field and field delimiter, which is inserted at the passed field element number.	183
<code>isEmpty()</code>	Determines whether the passed string is empty.	183
<code>length()</code>	Returns the length of a passed string.	183
<code>removeAt()</code>	Deletes a field from passed buffer at a specific element position.	184
<code>squeeze()</code>	Creates a string where all repeated white spaces in the passed string buffer are reduced to single spaces.	184
<code>substring()</code>	Returns a portion of the passed string.	184
<code>toString()</code>	Returns a string representation of the passed parameter.	185
<code>trim()</code>	Eliminates any leading and trailing spaces from the passed string.	185

## charAt()

Returns a single character that is located at the passed offset position within the passed string.

**Syntax** `String.charAt(Buffer, Offset);`

**Parameters** *Buffer* A buffer containing the source string.  
*Offset* The offset position within the source buffer.

**Returns** A single character located at the offset position.

**Example** The following code fragment returns the character "c":  

```
var theChar1 = String.charAt("abcdef", 3);
```

  
The following code fragment returns a null string:  

```
var theChar2 = String.charAt("abcdef", 8);
```

## compare()

Compares strings where ranking is performed based on the ASCII value of each character within the string.

**Syntax** `String.compare(String1, String2);`

**Parameters** *String1* The first string to compare.  
*String2* The second string to compare.

**Returns**

- 0: The strings are identical.
- -1: The first string is less than the second.
- 1: The second string is less than the first.

**Example** The following code fragment returns "0":  

```
var theRes1 = String.compare("ABC", "ABC");
```

  
The following code fragment returns "1":  

```
var theRes2 = String.compare("abc", "ABC");
```

  
The following code fragment returns "-1":  

```
var theRes3 = String.compare("ABC", "abc");
```

## elementAt()

Locates a single element within the passed string buffer.

**Syntax** `String.elementAt(Buffer, Element, Delimiter);`

**Parameters**

<i>Buffer</i>	A string buffer that contains delimited fields.
<i>Element</i>	A numeric value set to the desired field number. If the passed value is less than 0, then the first field is assumed. If it is greater than maximum number of elements, then the last field is assumed.
<i>Delimiter</i>	The character(s) used to delimit fields.

**Returns** The requested field.

**Example** The following code fragment returns "transformation":

```
var Field = String.elementAt("In an extraordinary
transformation of heat to light, Gibbon rested.", 4, " ");
```

## elements()

Counts how many times a delimiter occurs within a passed buffer to determine the number of elements in the buffer.

**Syntax** `String.elements(Buffer, Delimiter);`

**Parameters**

<i>Buffer</i>	A string buffer that contains delimited fields.
<i>Delimiter</i>	The character(s) used to delimit fields.

**Returns** The total count of *Delimiter* within *Buffer*.

**Example** The following code fragment returns "20":

```
var howMany = String.elements("This descent from unity into
multiplicity recalled Constantine's timid policy of 'dividing
whatever is united', but its effects were far different", " ");
```

## find()

Searches for the first occurrence of the passed substring within the passed buffer.

**Syntax** `String.find(Buffer, SubString);`

**Parameters**

<i>Buffer</i>	The source string buffer.
<i>SubString</i>	The substring to search for within the passed buffer.

**Returns:**

- Offset value of first occurrence of *SubString* (0-n).
- -1 if *SubString* is not found.

**Example** The following code fragment returns "2":

```
var theFirst = String.find( "Waterloo", "ter" );
```

## format()

Formats the passed numeric value as a string.

**Syntax** `String.format(FormatString, Value);`

**Parameters** *FormatString* Specifies the format of the string, which is configured as follows:

`%[width] [.precision] type`

- *width*: Optional. When specified, it specifies the minimum number of characters that must be returned in the string.
- *.precision*: Optional. When specified, it specifies the required decimal precision that is set based on the setting of *type*.
- *type*: Required. The *type* argument can have one of the following values:

- *d*: The source is treated as a positive or negative integer value in the form of `[-]n`, where *n* is one or more decimal digits.

If *.precision* is specified, then the output value is padded on the left side with up to the *.precision* number of zeroes.

- *f*: The source is treated as a positive or negative floating point value in the form of `[-]n.n`, where *n* is one or more decimal digits.

If *.precision* is specified, then it is used to set the number of digits after the decimal point, with at least one digit appearing before the decimal point. The default precision is 6. If 0 or a null value is specified after the decimal point, then the decimal component is truncated.

- *s*: The source is treated as a string.

In this case, the *width* argument can be used to specify the minimum string size, and the *.precision* argument can be used to specify the maximum string size.

*Value* The value to be formatted as a string.

**Returns** A formatted string.

**Example** The following code fragment returns "BlackBerry rules!":

```
var s5 = String.format("BlackBerry %s", "rules!");
```

The following code fragment returns " 98.765%" (with eight preceding blank spaces):

```
var s6 = String.format("%10.3F%", 98.7654);
```

## insertAt()

Creates a new string from the passed buffer that includes the passed field and field delimiter, which is inserted at the passed field element number.

**Syntax** `String.insertAt(Buffer, Field, Element, Delimiter);`

**Parameters** *Buffer* A string buffer that contains delimited fields.

*Field* The string field to insert.

*Element* A numeric element (0-*n*) where *Field* is to be inserted. If the passed value is less than 0, then 0 is used. If it is greater than the maximum number of elements, then *Field* is appended to the end of buffer.

*Delimiter* The delimiter character to be inserted after *Field*.

**Returns** The resulting string.

**Example** The following code fragment returns "1|99|2|":

```
var nStr = String.insertAt("1|2|", "99", 1, "|");
```

## isEmpty()

Determines whether the passed string is empty.

**Syntax** `String.isEmpty(Buffer);`

**Parameters** *Buffer* The source string buffer.

**Returns**

- Boolean true if the string is empty.
- Boolean false if it is not empty.

**Example** The following code fragment returns true:

```
var NULLString = "";
var IsNULL = String.isEmpty(NULLString);
```

## length()

Returns the length of a passed string.

**Syntax** `String.length(Buffer);`

**Parameters** *Buffer* The source string buffer.

**Returns** The string length (0-*n*).

**Example** The following code fragment returns 6:

```
var theLength = String.length("yellow");
```

The following code fragment returns 0:

```
var theLength = String.length("");
```

**removeAt()**

Removes a field from the passed buffer at a specific element position.

**Syntax** `String.removeAt(Buffer, Element, Delimiter);`

**Parameters**

<i>Buffer</i>	The source string buffer.
<i>Element</i>	The field element to remove from the buffer.
<i>Delimiter</i>	The character delimiter used to separate fields.

**Returns** *Buffer*, without the requested element.

**Example** The following code fragment returns "1|3|":

```
var Res = String.removeAt("1|2|3|", 1, "|");
```

**squeeze()**

Creates a string where all repeat white spaces in the passed string buffer are reduced to single spaces.

**Syntax** `String.squeeze(Buffer);`

**Parameters** *Buffer* The source string buffer.

**Returns** *Buffer* with "squeezed" spaces.

**Example** The following code fragment returns "My BlackBerry is cool!":

```
var SqzMe = String.squeeze("My   BlackBerry   is   cool!");
```

**subString()**

Returns a portion of the passed string.

**Syntax** `String.subString(Buffer, Start, Size);`

**Parameters**

<i>Buffer</i>	The source string buffer.
<i>Start</i>	The position of the first character of the substring in the source (0- <i>n</i> ).
<i>Size</i>	The total number of characters to extract.

**Returns** The requested substring.

**Example** The following code fragment returns "Berry":

```
var SS = String.subString("BlackBerry", 5, 5);
```



## toString()

Returns a string representation of the passed parameter.

**Syntax**      `String.toString(Value);`

**Parameters**   *Value*      Any value.

**Returns**      A string.

**Example**      The following code fragment returns a string with the value "98.76":

```
var theString = String.toString(98.76);
```

## trim()

Eliminates any leading and trailing spaces from the passed string.

**Syntax**      `String.trim(Buffer);`

**Parameters**   *Buffer*      The source string buffer.

**Returns**      The string buffer without leading and trailing spaces.

**Example**      The following code fragment returns "My BlackBerry is cool!":

```
var isTrimmed = String.trim(" My BlackBerry is cool! ");
```

## URL

The URL library contains 14 functions that manipulate URLs. For example, you can extract the various portions of the URL, escape and unescape special characters in the URL, determine the referring URL, and so on.

A URL is composed of some or all of the following components:

scheme://host:port/path;parameters\$query#fragment.

Function	Description	See page
<code>escapeString()</code>	Returns a string where special characters are changed into hexadecimal escape sequences.	186
<code>getBase()</code>	Returns the absolute URL (without the fragment) of the current WMLScript.	186
<code>getFragment()</code>	Returns the fragment portion of the passed URL.	186
<code>getHost()</code>	Returns the host that is specified within the passed URL.	187
<code>getParameters()</code>	Returns the parameters within the last path segment of the passed URL.	187
<code>getPath()</code>	Returns the path that is specified within the passed URL.	187
<code>getPort()</code>	Returns the port specified within the passed URL.	188
<code>getQuery()</code>	Returns the query portion of the passed URL.	188
<code>getReferer()</code>	Returns the smallest relative URL for the page, deck, or script that called the current script.	188
<code>getScheme()</code>	Returns the scheme within the passed URL.	189
<code>isValid()</code>	Validates the syntax of the passed URL.	189
<code>loadString()</code>	Returns the content that the passed absolute URL and content type refer.	189
<code>resolve()</code>	Combines the passed base and relative URLs to return an absolute URL.	190
<code>unescapeString()</code>	Returns a string where escaped characters have been restored to original form.	190

## escapeString()

Returns a string where special characters are changed into hexadecimal escape sequences.

The following special characters should be converted:

- reserved characters: semicolon (;), slash (/), question mark (?), colon (:), at symbol (@), ampersand (&), equal sign (=), plus sign (+), and dollar sign (\$)
- unwise characters: left and right brace ( { } ), vertical slash (|), backslash (\), caret (^), left and right brackets ( [ ] ), and apostrophe (')
- delimiters: greater than sign(>), less than sign(<), number sign (#), percent symbol (%), and quotation marks (")

The resulting escaped characters are as follows:

- control characters (ASCII %00 to %1F and %7F)
- space (ASCII %20)
- upper range (ASCII %8F to %FF)

**Syntax** `URL.escapeString(URL);`

**Parameters** *URL* A string buffer containing the unescaped URL.

**Returns** A string buffer containing the escaped URL.

**Example** The following code fragment returns "http%3a%2f%2frim.com%2f":

```
URL.escapeString("http://rim.com/");
```

## getBase()

Returns the absolute URL (without the fragment) of the current WMLScript.

**Syntax** `URL.getBase();`

**Returns** The absolute URL.

**Example** The following code fragment returns "http://rim.com/script.wmls":

```
var theURL = "http://rim.com/script.wmls#frag"
var absURL = URL.getBase();
```

## getFragment()

Returns the fragment portion of the passed URL.

**Syntax** `URL.getFragment(URL);`

**Parameters** *URL* The passed URL.

**Returns** The URL fragment.

**Example** The following code fragment returns "frag":

```
var theURL = "http://rim.com/script.wmls#frag"
var theFrag = URL.getFragment(theURL);
```

## getHost()

Returns the host that is specified within the passed URL.

**Syntax** `URL.getHost(URL);`

**Parameters** *URL* The passed URL.

**Returns** The host component of the URL.

**Example** The following code fragment returns "www.rim.com":

```
var theURL = "http://www.rim.com/script.wmls";
var theHost = URL.getHost(theURL);
```

The following code fragment returns a null string, as the passed URL contains no host component:

```
var theURL = "script.wmls";
var theHost = URL.getHost(theURL);
```

## getParameters()

Returns the parameters within the last path segment of the passed URL.

**Syntax** `URL.getParameters(URL);`

**Parameters** *URL* The passed URL.

**Returns** The parameters of the URL.

**Example** The following code fragment returns "param1;param2":

```
var theURL = "http://rim.com/sample.php;param1;param2";
var parms = URL.getParameters(theURL);
```

The following code fragment returns a null string, as the passed URL contains no parameters:

```
var theURL = "http://www.rim.com/script.wmls";
var parms = URL.getParameters(theURL);
```

## getPath()

Returns the path that is specified within the passed URL.

**Syntax** `URL.getPath(URL);`

**Parameters** *URL* The passed URL.

**Returns** The path component.

**Example** The following code fragment returns "test/sample.php":

```
var theURL = "http://rim.com/test/sample.php";
var thePath = URL.getPath(theURL);
```

The following code fragment returns a null string, as the passed URL contains no path:

```
theURL = "http://rim.com/";
thePath = URL.getPath(theURL);
```

## getPort()

Returns the port number that is specified within the passed URL.

**Syntax** `URL.getPort(URL);`

**Parameters** *URL* The passed URL.

**Returns** The port component.

**Example** The following code fragment returns "80":

```
var theURL = "http://www.rim.com:80";
var thePort = URL.getPort(theURL);
```

The following code fragment returns a null string, as the passed URL contains no port:

```
theURL = "http://www.rim.com";
thePort = URL.getPort(theURL);
```

## getQuery()

Returns the query portion of the passed URL.

**Syntax** `URL.Query(URL);`

**Parameters** *URL* The passed URL.

**Returns** The query portion of the URL.

**Example** The following code fragment returns "4884":

```
var theURL = "http://rim.com/sample.php?id=4884";
var theQuery = URL.getQuery(theURL);
```

The following code fragment returns a null string, as the passed URL contains no query component:

```
theQuery = URL.getQuery("http://www.rim.com");
```

## getReferer()

Returns the smallest relative URL for the page, deck, or script that called the current script.

**Syntax** `URL.getReferer();`

**Returns** The referring URL.

**Example** The following code fragment might return the full URL or something relative such as "mydeck.wml". If no referrer exists, it returns a null string:

```
var whoCalled = URL.getReferer();
```

## getScheme()

Returns the scheme within the passed URL.

**Syntax** `URL.getScheme(URL);`

**Parameters** *URL* The passed URL.

**Returns** The scheme of the URL.

**Example** The following code fragment returns "http":

```
var theURL = "http://www.rim.com/";
var theScheme = URL.getScheme(theURL);
```

The following code fragment returns a null string, as the passed URL contains no scheme component:

```
var theURL = "www.rim.com/";
var theScheme = URL.getScheme(theURL);
```

## isValid()

Validates the syntax of the passed URL.

**Syntax** `URL.isValid(URL);`

**Parameters** *URL* The passed URL.

**Returns**

- Boolean true if the syntax is correct.
- Boolean false if the syntax is not correct.

**Example** The following code fragment returns "true":

```
var theURL = "http://www.rim.com/";
var isOkay = URL.isValid(theURL);
```

The following code fragment returns "false":

```
var theURL = "http://www.rim.com/";
var isOkay = URL.isValid(theURL);
```

## loadString()

Returns the content that the passed absolute URL and content type refer.

**Syntax** `URL.loadString(URL, ContentType);`

**Parameters**

<i>URL</i>	A string that contains an absolute URL.
<i>ContentType</i>	A string that contains the accepted content type that must be prefixed with "text/".

**Returns** A string buffer that contains the requested page, deck, or script.

**Example** The following code fragment returns the page contents:

```
var theURL = "http://www.rim.com/index.shtml";
var theCT = "text/plain";
var theContent = URL.loadString(theURL, theCT);
```

## resolve()

Combines the passed base and relative URLs to return an absolute URL.

**Syntax** `URL.resolve(baseURL, relURL);`

**Parameters** *baseURL* The base portion of the passed URL (for example, "http://www.rim.com/").  
*relURL* The relative path to a page, deck, or script (for example, "index.shtml").

**Returns** The resulting absolute URL.

**Example** The following code fragment returns "http://www.rim.com/index.shtml":

```
var baseURL = "http://www.rim.com/";
var relURL = "index.shtml";
var absURL = URL.resolve(baseURL, relURL);
```

## unescapeString()

Returns a string where escaped characters have been restored to original form.

**Syntax** `URL.unescapeString(escURL);`

**Parameters** *escURL* An escaped URL.

**Returns** An unescaped URL.

**Example** The following code fragment returns "http://rim.com/":

```
var escURL = "http%3a%2f%2frim.com%2f";
var newURL = URL.unescapeString(escURL);
```

## Browser

The Browser library contains seven functions that can be used to access the information that is associated with the WML cards. For example, you can return to a previously viewed card, go to a new card, refresh a card, and so on.

Function	Description	See page
<code>getCurrentCard()</code>	Returns the smallest relative URL of the current card that the browser is processing. If the current card has a different base than the current script, this function returns the absolute URL of the card.	191
<code>getVar()</code>	Returns the value of the passed variable name within the current browser context.	191
<code>go()</code>	Navigates the browser to the given URL.	191
<code>newContext()</code>	Resets the browser context and clears all variables.	191
<code>prev()</code>	Navigates the browser to the previous card.	192
<code>refresh()</code>	Refreshes the current page by pulling it from the server.	192
<code>setVar()</code>	Specifies the value of a variable.	192

## getCurrentCard()

Returns the smallest relative URL of the current card that the browser is processing. If the current card has a different base than the current script, this function returns the absolute URL of the card.

**Syntax**      `Browser.getCurrentCard();`

**Returns**      A relative or absolute URL.

**Example**      The following code fragment returns the current card being processed by the browser:

```
var relURL = Browser.getCurrentCard();
```

## getVar()

Returns the value of the passed variable name within the current browser context.

**Syntax**      `Browser.getVar(strName);`

**Parameters**   *strName*   The name of the variable for which the value is to be returned.

**Returns**      String value or invalid if not found.

**Example**      The following code fragment returns the variable name (for example, "revers"):

```
var varVal = Browser.getVar("userid");
```

## go()

Navigates the browser to the given URL.

**Syntax**      `Browser.go(navURL);`

**Parameters**   *navURL*   A relative or absolute URL to which the browser is pointed.

**Returns**      A null string.

**Example**      The following code fragment navigates the browser to a relative URL:

```
var ret = Browser.go("newpage.wml");
```

The following code fragment navigates the browser to an absolute URL:

```
var ret = Browser.go("http://www.xyzyzy.com/");
```

## newContext()

Resets the browser context and clears all variables.

**Syntax**      `Browser.newContext();`

**Returns**      A null string.

**Example**      The following code fragment resets the browser context:

```
var ret = Browser.newContext();
```

## prev()

Navigates the browser to the previous card.

**Syntax** `Browser.prev();`

**Returns** A null string.

**Example** The following code fragment navigates the browser to the previous card:

```
var ret = Browser.prev();
```

## refresh()

Refreshes the current page by pulling it from the server.

**Syntax** `Browser.refresh();`

**Parameters** None.

**Returns** A null string.

**Example** The following code fragment refreshes the current page:

```
var ret = Browser.refresh();
```

## setVar()

Specifies the value of a variable.

**Syntax** `Browser.setVar(varName, varValue);`

**Parameters** *varName* The name of the variable for which the value is to be specified.

*varValue* The value to assign to *varName*.

**Returns**

- Boolean true on success.
- Boolean false on failure.

**Example** The following code fragment sets the variable *password* to a value of "xyzyz":

```
var isSet = Browser.setVar("password", "xyzyz");
```



# Scripting Basics

Reserved words

Statements

Operators and expressions

## Reserved words

The following reserved words in WMLScript and JavaScript cannot be used to name functions, variables, methods, or objects:

- abstract
- access
- agent
- Boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- debugger
- default
- delete
- div
- do
- domain
- double
- else
- equiv
- enum
- export
- extends
- extern
- false
- final
- finally
- float
- for
- function
- goto
- header
- http
- if
- implements
- import
- in
- instanceof
- int
- interface
- invalid
- long
- meta
- name
- native
- new
- null
- package
- path
- private
- protected
- public
- return
- short
- sizeof
- static
- struct
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- true
- try
- typeof
- url
- use
- user
- var
- void
- volatile
- while
- with

## Statements

The BlackBerry Browser supports the following WMLScript and JavaScript statements:

Statement	Description	Example	Supported by	
			WMLScript	JavaScript
break	<p>Stops a for or while loop statement. All processing is immediately stopped inside the loop and the loop is exited. The program continues with the first line of code (if any) after the terminated loop.</p> <p>Using a break statement outside of a for or while loop statement generates an error.</p>	<pre>var i = 0; while ( i &lt; 6 ) {   if ( i == 3 ) {     break;     i++;   }   return i*x; }</pre>	✓	✓
continue	<p>Stops a block of statements inside a for or while loop statement and redirects the program to another block of statements inside the loop.</p> <p>In a for loop, the program jumps to the for counting variable test expression.</p> <p>In a while loop, the program jumps to the while condition test.</p>	<pre>i = 0; n = 0; while ( i &lt; 5 ) {   i++;   if ( i == 3 )     continue;   n += i; }</pre>	✓	✓
do ... while	Runs one or more statements at least once, checking that a certain condition is met each time before repeating. If that condition is not met, then control moves to the statement immediately after the loop.	<pre>var i = 0; do {   document.write( i + "&lt;BR&gt;");   i+=2; } while( i&lt;20 );</pre>		✓
for	Repeats a block of statements as long as a stated test condition, based upon a counting mechanism, remains true.	<pre>for( i=0; i&lt;10; i++)   document.write( i + "&lt;BR&gt;");</pre>	✓	✓
for ... in	Iterates a declared variable over every property in a specified object.	<pre>var i; for( i in mimeTypeArray )   document.write( i + "&lt;BR&gt;");</pre>		✓
if ... else	<p>Evaluates a specified expression contained in the if statement to determine if it is true or false. If true, a block of statements associated in the if statement is run. If false, the if statement is immediately exited, unless an optional else clause exists.</p> <p>Use the optional else clause to run a block of statements if the specified condition is false. Note that you cannot provide a test expression for the else statement.</p>	<pre>if( calcaverage (x,y,z ) &lt; 10 )   document.write("The average is     less than 10."); else   document.write("The average is     10 or more.");</pre>	✓	✓
return	Specifies the value to be returned by a function and performs the act of returning that value to where the function was called from.	<pre>function square( x ) {   return x * x; }</pre>	✓	✓

Statement	Description	Example	Supported by	
			WMLScript	JavaScript
switch	Tests an expression against a number of case options , then runs the statements associated with the first case that matches the expression. If no match is found, the program looks for a set of default statements to run, and if these are not found, it carries on with the statement immediately following switch.	<pre>switch (modelNumber ) {   case "6700" :     document.write ("monochrome");     break;   case "7700" :     document.write ("color");     break;   default :     document.write ("Sorry, " + i +       " is not a valid       model.&lt;BR&gt;"); }</pre>		✓
var	<p>Declares a variable. Outside a function it is optional. While a variable can be declared by assigning it a value, there are two cases in functions where using var is necessary:</p> <ul style="list-style-type: none"> <li>• If a global variable of the same name exists.</li> <li>• If recursive or multiple functions use variables of the same name.</li> </ul> <p>You can also declare more than one variable and, optionally, assign values at the same time.</p>	<pre>var i = 0; var num_hits = 0, cust_no = 0;</pre>	✓	✓
while	Repeats a block of statements as long as a stated test condition, based upon an evaluated expression, remains true.	<pre>var i = 0; while( i&lt;11 ) {   document.write (i + &lt;BR&gt;);   i++; }</pre>	✓	✓
with	A statement that establishes the default object for a set of statements. Within the set of statements, any property references that do not specify an object are assumed to be for the default object.	<pre>with(displayType) {   if( model &lt; 7000 )     document.write("monochrome");   else     document.write("color"); }</pre>	✓	✓

## Operators and expressions

The BlackBerry Browser supports the following WMLScript and JavaScript operators and expressions:

Operator type	Symbol	Description	Example	Supported by	
				WMLScript	JavaScript
Arithmetic	+	Addition. Returns the sum of two numerical values.	if x = 5 and y = 2 then x + y returns 7	✓	✓
	-	Subtraction. Subtracts one numerical value from another.	if x = 5 and y = 2 then x - y returns 3	✓	✓
	*	Multiplication. Returns the product of two numerical values.	if x = 5 and y = 2 then x * y returns 10	✓	✓
	/	Division. Divides one numerical value by another.	if x = 5 and y = 2 then x / y returns 2.5	✓	✓
	div	Integer division. Divides one numerical value by another, and rounds the result down to the nearest whole number.	if x = 5 and y = 2 then x div y returns 2	✓	
	%	Remainder. Returns the integer remaining when one operand is divided by another.	if x = 5 and y = 2 then x % y returns 1	✓	✓
Unary	+	Positive. Precedes an operand to indicate it is greater than zero.	if x = 5 then +x returns 5	✓	✓
	-	Negation. Precedes an operand and negates it.	if x = 5 then -x returns -5	✓	✓
	++	Increment. When positioned <ul style="list-style-type: none"> <li>after the operand, it returns the value before incrementing.</li> <li>before the operand, it increments before returning the value.</li> </ul>	if x = 5: <ul style="list-style-type: none"> <li>y = x++ sets y to 5, then increases x to 6</li> <li>y = ++x increases x to 6, then sets y to 6</li> </ul>	✓	✓
	--	Decrement. When positioned <ul style="list-style-type: none"> <li>after the operand, it returns the value before decrementing.</li> <li>before the operand, it decrements before returning the value.</li> </ul>	if x = 5: <ul style="list-style-type: none"> <li>y = x-- sets y to 5, then decreases x to 4</li> <li>y = --x decreases x to 4, then sets y to 4</li> </ul>	✓	✓
	~	Bitwise NOT. Flips the bit values of its operand.	~ 9 returns 6 (1001 becomes 110)	✓	✓

Operator type	Symbol	Description	Example	Supported by	
				WMLScript	JavaScript
Comparison	==	Equal. Returns true if the values of the left and right operands are identical.	if x = 5 and y = 2 then x == y returns false	✓	✓
	>	Greater than. Returns true if the value of the left operand is higher than the value of the right operand.	if x = 5 and y = 2 then x > y returns true	✓	✓
	<	Less than. Returns true if the value of the left operand is lower than the value of the right operand.	if x = 5 and y = 2 then x < y returns false	✓	✓
	>=	Greater than or equal. Returns true if the value of the left operand is equal to or greater than the value of the right operand.	if x = 5 and y = 2 then x >= y returns true	✓	✓
	<=	Less than or equal. Returns true if the value of the left operand is equal to or less than the value of the right operand.	if x = 5 and y = 2 then x <= y returns false	✓	✓
	!=	Not equal. Returns true if the values of the left and right operands are not equal.	if x = 5 and y = 2 then x != returns true	✓	✓
Bitwise	<<	Left shift. Shifts the digits of the binary representation of the first operand to the left by the number of places specified by the second operand. The spaces that are created to the right are filled in by zeros, and any digits falling off the left are discarded.	9 << 2 returns 36 (1001 becomes 100100)	✓	✓
	>>	Sign-propagating right shift. Shifts the digits of the binary representation of the first operand to the right by the number of places specified by the second operand, discarding any numbers shifted off to the right. Copies of the left most bit are shifted in from the left.	9 >> 2 returns 2 (1001 becomes 10) -9 >> 2 returns -3, because the sign is preserved	✓	✓
	>>>	Right shift with zero fill. Shifts the digits of the binary representation of the first operand to the right by the number of places specified by the second operand, discarding any digits shifted off to the right and adding zeros to the left.	19 >>> 2 returns 4 (10011 becomes 100)	✓	✓
	&	AND. Returns "1" for each bit position where the corresponding bits of its operands is "1".	15 & 9 would return 9 (1111 & 1001 = 1001)	✓	✓
		OR. Returns "1" for each bit position where one or both of the corresponding bits of its operands is "1".	15   9 would return 15 (1111   1001 = 1111)	✓	✓
	^	Exclusive OR. Returns "1" for each bit position where only one (not both) of the corresponding operands is "1".	15 ^ 9 would return 6 (1111 ^ 1001 = 110)	✓	✓

Operator type	Symbol	Description	Example	Supported by	
				WMLScript	JavaScript
Assignment	=	Assignment. Assigns a literal or numerical value on the right to the variable on the left.	<code>x = y</code> (sets the value of x to the value of y)	✓	✓
	+=	Addition and assignment. Adds the operand to the variable, and sets the variable to the result.	if x = 5 and y = 2 then <code>x += y</code> returns 7 (equivalent to <code>x = x + y</code> )	✓	✓
	-=	Subtraction and assignment. Subtracts the operand from the variable, and sets the variable to the result.	if x = 5 and y = 2 then <code>x -= y</code> returns 3 (equivalent to <code>x = x - y</code> )	✓	✓
	*=	Multiplication and assignment. Multiplies the variable and the operand, and sets the variable to the result.	if x = 5 and y = 2, then <code>x *= y</code> returns 10 (equivalent to <code>x = x * y</code> )	✓	✓
	/=	Division and assignment. Divides the variable by the operand, and sets the variable to the result.	if x = 5 and y = 2, then <code>x /= y</code> returns 2.5 (equivalent to <code>x = x / y</code> )	✓	✓
	div=	Integer division and assignment. Divides the variable by the operand, rounds the result down to the nearest whole number, and sets the variable to the result.	if x = 5 and y = 2, then <code>x div= y</code> returns 2 (equivalent to <code>x = x div y</code> )	✓	
	%=	Remainder and assignment. Determines the integer remainder when the variable is divided by the operand, and sets the variable to the remainder value.	if x = 5 and y = 2, then <code>x %= y</code> returns 1 (equivalent to <code>x = x % y</code> )	✓	✓
	<<=	Bitwise left shift and assignment. Shifts the digits of the bitwise variable left by the number of positions specified by the operand, and sets the variable to the result. This operation is not permissible if the initial value of the variable is negative.	if x = 9 and y = 2, then <code>x &lt;&lt;= y</code> returns 36 (equivalent to <code>x = x &lt;&lt; y</code> )	✓	✓
	>>=	Bitwise right shift and assignment. Shifts the digits of the bitwise variable right by the number of positions specified by the operand, and sets the variable to the result. This operation is not permissible if the initial value of the variable is negative.	if x = 9 and y = 2, then <code>x &gt;&gt;= y</code> returns 2 (equivalent to <code>x = x &gt;&gt; y</code> )	✓	✓
	>>>=	Bitwise right shift zero fill and assignment. Shifts the digits of the bitwise variable right by the number of spaces specified by the operand, and sets the value of the variable to the result.	if x = 19 and y = 2, then <code>x &gt;&gt;&gt;= y</code> returns 4 (equivalent to <code>x = x &gt;&gt;&gt; y</code> )	✓	✓
	&=	Bitwise AND and assignment. Performs a bitwise AND operation on the variable and the operand, and sets the value of the variable to the result.	if x = 15 and y = 9, then <code>x &amp;= y</code> returns 9 (equivalent to <code>x = x &amp; y</code> )	✓	✓
	=	Bitwise OR and assignment. Performs a bitwise OR operation on the variable and the operand, and sets the value of the variable to the result.	if x = 15 and y = 9, then <code>x  = y</code> returns 15 (equivalent to <code>x = x   y</code> )	✓	✓
	^=	Bitwise exclusive OR and assignment. Performs a bitwise exclusive OR operation on the variable and the operand, and sets the value of the variable to the result.	if x = 15 and y = 9, then <code>x ^= y</code> returns 6 (equivalent to <code>x = x ^ y</code> )	✓	✓

Operator type	Symbol	Description	Example	Supported by	
				WMLScript	JavaScript
Logical	&&	AND. Returns a Boolean true if both of the given expressions are true.	if x = 5 and y = 2 then (y < x) && (y > (x >>2)) returns true	✓	✓
		OR. Returns a Boolean true if one of the given expressions are true.	if x = 5 and y = 2 then (x < y)    (y > (x >>2)) returns true	✓	✓
	!	NOT. Returns Boolean true if the given expression is false, and returns Boolean false if the given expression is true.	if x = 5 and y = 2 then ! (x < y) returns true, and (y < x) returns false	✓	✓
String	+	Concatenate. Joins two string values together.	"new" + "_string" returns "new_string"	✓	✓
	+=	Concatenate and assignment. Adds a string to the string variable, and sets the variable to the result.	If mystring = "new" then mystring += "_string" returns "new_string"	✓	✓
Special	?:	Conditional operator that takes three operands and is used to replace simple if statements. The first operand is a condition that evaluates to true or false, the second is to be returned on true, and the third to be returned on false.	var IsOkay = ( Value == "Food" ) ? 1 : 0;	✓	✓
	,	Comma. Used to include multiple expressions where only one is required, such as in a for loop. It evaluates both its operands and returns the value of the second.	for (el = 0, id = 100; el < 10; el++, id+=10)	✓	✓
	typeof	Returns the data type of the given variable as one of number, string, Boolean, object, or function.	var result = typeof data;	✓	✓
	isvalid	Tests whether an expression is valid. It returns true if the passed expression is valid, or else it returns false.	var IsOk_1 = isvalid (99/0);	✓	
	new	Creates an instance of a user-defined object type or of one of the following built-in object types: array, Boolean, date, function, math, number, or string. Uses the following syntax:  <i>objectName</i> = new <i>objectType</i> ( <i>param1</i> [, <i>param2</i> ] ... [, <i>paramM</i> ] )	formsArray = new array ( <i>form1</i> , <i>form2</i> , <i>form3</i> )		✓
	delete	Deletes an object, an object property, or a specified element in an array, returning true if the operation is possible, and false if it is not.	x=42 delete x		✓
	void	Specifies an expression to be evaluated without returning a value.	void (document.form. submit())		✓







